# 构建 ADOBE® AIR® 应用程序

Adobe

## 法律声明

# 目录

**第 13 章 : 对 AIR 应用程序进行签名**



**第 14 章 : AIR 应用程序描述符文件**



**第 15 章 : 设备配置文件**



**第 16 章 : AIR.SWF 浏览器内 API**



**第 17 章 : 更新 AIR 应用程序**



**第 18 章 : 查看源代码**



**第 19 章 : 使用 AIR HTML 内部检查器进行调试**

**第 20 章 : 本地化 AIR 应用程序**



**第 21 章 : 路径环境变量**

# 第 1 章：关于 Adobe AIR

Adobe® AIR® 是一个跨操作系统的多屏幕运行时，通过它可以利用您的 Web 开发技能来构建丰富 Internet 应用程序 (RIA)，并将其部署到桌面和移动设备上。可以使用 Adobe® Flex 和 Adobe® Flash®（基于 SWF）通过 ActionScript 3.0 构建桌面、电视和移动 AIR 应用程序。桌面 AIR 应用程序也可以使用 HTML、 JavaScript® 和 Ajax （基于 HTML）进行构建。

在 Adobe AIR 开发人员中心 (http://www.adobe.com/cn/devnet/air/) 上可以找到有关 Adobe AIR 入门和使用的详细信息。

通过 AIR，您可以在熟悉的环境中工作，可以利用您认为最方便的工具和方法。由于它支持 Flash、 Flex、 HTML、 JavaScript 和 Ajax，您可以创造满足您需要的可能的最佳体验。

例如：可以使用以下技术之一或其某一组合开发应用程序：

- Flash/Flex/ActionScript
- HTML/JavaScript/CSS/Ajax

用户与 AIR 应用程序交互的方式和他们与本机应用程序交互的方式相同。在用户计算机或设备上安装此运行时之后，即可像任何其他桌面应用程序一样安装和运行 AIR 应用程序。（在 iOS 上，未单独安装 AIR 运行时；每个 iOS AIR 应用程序都是独立应用程序。）

此运行时通过在不同桌面间确保一致的功能和交互来提供用于部署应用程序的一致性跨操作系统平台和框架，从而消除跨浏览器测试。不是针对特定操作系统进行开发，而是以此运行时为目标，它具有以下优点：

- 针对 AIR 开发的应用程序可以在多个操作系统上运行，同时不需要进行额外的工作。此运行时确保在由 AIR 支持的所有操作系统上进行一致并可预知的呈现和交互。
- 利用现有的 Web 技术和设计模式可以更快地构建应用程序。无需学习传统的桌面开发技术或复杂的本机代码，您即可将基于 Web 的应用程序扩展到桌面。
- 与使用诸如 C 和 C++ 之类的较低级别的语言相比，使用此运行时可以更轻松地开发应用程序。无需管理特定于每个操作系统的复杂的低级别 API。

当针对 AIR 开发应用程序时，可以利用一组丰富的框架和 API：

- 由此运行时提供的特定于 AIR 的 API 和 AIR 框架
- SWF 文件中使用的 ActionScript API 和 Flex 框架 （以及其他基于 ActionScript 的库和框架）
- HTML、 CSS 和 JavaScript
- 大多数 Ajax 框架
- Adobe AIR 的本机扩展，其提供 ActionScript API，该 API 提供对采用本机代码编程的特定于平台的功能的访问。本机扩展还会提供对旧本机代码以及提供更高性能的本机代码的访问。

AIR 在很大程度上改变了应用程序的创建、部署和使用方式。您获得了更富有创造性的控制能力，并可以将您的基于 Flash、 Flex、 HTML 和 Ajax 的应用程序扩展到桌面、移动设备和电视。

有关每个新 AIR 更新中所包含内容的信息，请参阅 Adobe AIR 发行说明 (http://www.adobe.com/go/learn_air_relnotes_cn)。

# 第 2 章：Adobe AIR 安装

利用 Adobe® AIR® 运行时，可以运行 AIR 应用程序。您可以通过以下方式安装运行时：

- 单独安装运行时 （不同时安装 AIR 应用程序）
- 首次通过网页安装 " 标志 " 安装 AIR 应用程序 （同时提示安装运行时）
- 创建安装应用程序和运行时的自定义安装程序。必须获得 Adobe 的批准才能以此方式分发 AIR 运行时。您可通过 Adobe 运行时许可页面请求批准。请注意， Adobe 不提供用于捆绑此类安装程序的工具。但是，有许多第三方安装程序工具包可用。
- 安装将 AIR 捆绑为捕获运行时的 AIR 应用程序。只有捆绑应用程序才使用捕获运行时。该运行时不用于运行其他 AIR 应用程序。捆绑运行时在 Android、 Mac 和 Windows 上是可选的。在 iOS 上，所有应用程序都包含捆绑的运行时。
- 设置 AIR 开发环境，如 AIR SDK、Adobe® Flash® Builder™ 或 Adobe Flex® SDK（包含 AIR 命令行开发工具）。SDK 中包含的运行时仅用于调试应用程序，不可用于运行已安装的 AIR 应用程序。

Adobe AIR: 系统要求 (http://www.adobe.com/cn/products/air/systemreqs/) 详细介绍了安装 AIR 和运行 AIR 应用程序的系统要求。

运行时安装程序和 AIR 应用程序安装程序在安装、更新或删除 AIR 应用程序或者 AIR 运行时本身时，将创建日志文件。可参考这些日志以帮助确定任何安装问题的原因。请参阅安装日志。

## 安装 Adobe AIR

要安装或更新运行时，用户必须对计算机具有管理权限。

**在 Windows 计算机上安装运行时**

1 从 http://get.adobe.com/cn/air 下载运行时安装文件。
2 双击运行时安装文件。
3 在安装窗口中，按照提示完成安装。

**在 Mac 计算机上安装运行时**

1 从 http://get.adobe.com/cn/air 下载运行时安装文件。
2 双击运行时安装文件。
3 在安装窗口中，按照提示完成安装。
4 如果安装程序显示 " 身份验证 "(Authenticate) 窗口，请输入 Mac OS 用户名和密码。

**在 Linux 计算机上安装运行时**

注：此时， Linux 上不支持 AIR 2.7 和更高版本。对 Linux 部署的 AIR 应用程序应该继续使用 AIR 2.6 SDK。

使用二进制安装程序：

1 安装二进制文件可在 http://kb2.adobe.com/cps/853/cpsid_85304.html 上找到并下载。
2 设置文件权限以便可执行安装应用程序。在命令行中您可以使用以下代码设置文件权限：

```
chmod +x AdobeAIRInstaller.bin
```

某些 Linux 版本允许您在通过上下文菜单打开的 " 属性 " 对话框上设置文件权限。

**3** 从命令行中或通过双击运行时安装文件运行安装程序。

**4** 在安装窗口中，按照提示完成安装。

Adobe AIR 以本机软件包形式进行安装。即，在基于 rpm 发行版中以 rpm 的形式安装，在 Debian 发行版中以 deb 的形式安装。当前， AIR 不支持任何其他软件包格式。

使用软件包安装程序：

**1** AIR 包文件可在 http://kb2.adobe.com/cps/853/cpsid_85304.html 上找到并下载。根据您的系统支持的软件包格式，下载 rpm 或 Debian 软件包。

**2** 如果需要，双击 AIR 包文件以安装此软件包。

您也可以通过命令行安装：

**a** 在 Debian 系统中：

```
sudo dpkg -i <path to the package>/adobeair-2.0.0.xxxxx.deb
```

**b** 在基于 rpm 的系统中：

```
sudo rpm -i <path to the package>/adobeair-2.0.0-xxxxx.i386.rpm
```

或者，如果您更新现有版本 （AIR 1.5.3 或更高版本）：

```
sudo rpm -U <path to the package>/adobeair-2.0.0-xxxxx.i386.rpm
```

要安装 AIR 2 和 AIR 应用程序，您必须对计算机具有管理员权限。

Adobe AIR 将安装到以下位置：/opt/Adobe AIR/Versions/1.0

AIR 注册 mime 类型 “application/vnd.adobe.air-application-installer-package+zip”，这表示 .air 文件属于此 mime 类型，因此在 AIR 运行时中注册。

### 在 **Android** 设备上安装运行时

您可以通过 Android Market 安装 AIR 运行时的最新发布版。

您可以通过网页上的链接或使用 ADT -installRuntime 命令来安装 AIR 运行时的开发版。一次只能安装一个版本的 AIR 运行时 ；不能同时安装发布版和开发版。

有关详细信息，请参阅 第 133 页的 “ADT installRuntime 命令 ”。

### 在 **iOS** 设备上安装运行时

必要的 AIR 运行时代码与针对 iPhone、 iTouch 和 iPad 设备创建的每个应用程序捆绑在一起。您不必安装单独的运行时组件。

**更多帮助主题**

第 60 页的 “AIR for iOS”

# 删除 Adobe AIR

安装运行时之后，可以通过以下步骤将其删除。

### 在 **Windows** 计算机上删除运行时

**1** 在 Windows 的 “ 开始 ” 菜单中，选择 “ 设置 ”>“ 控制面板 ”。

**2** 打开 “ 程序 ”、 “ 程序和功能 ” 或 “ 添加或删除程序 ” 控制面板 （取决于您运行的 Windows 版本）。

3 选择 “Adobe AIR” 以删除运行时。

4 单击 “ 更改 / 删除 ” 按钮。

在 **Mac** 计算机上删除运行时

- 双击 “Adobe AIR Uninstaller”，其位于 /Applications/Utilities 文件夹中。

在 **Linux** 计算机上删除运行时

请执行下列操作之一：

- 从 “ 应用程序 ” 菜单中选择 “Adobe AIR Uninstaller” 命令。
- 运行 AIR 安装程序二进制文件，并加入 -uninstall 选项
- 用包管理器删除 AIR 包 （adobeair 和 adobecerts）。

从 **Android** 设备删除运行时

1 在设备上打开 “ 设置 ” 应用程序。

2 点击 “ 应用程序 ”>“ 管理应用程序 ” 下的 Adobe AIR 条目。

3 点击 “ 卸载 ” 按钮。

您还可以使用 ADT -uninstallRuntime 命令。有关详细信息，请参阅 第 134 页的 “ADT uninstallRuntime 命令 ”。

删除捆绑的运行时

要删除捕获捆绑的运行时，必须删除其随之一起安装的应用程序。请注意，捕获运行时仅用于运行安装应用程序。

# 安装和运行 AIR 范例应用程序

要安装或更新 AIR 应用程序，用户必须对计算机具有管理权限。

某些范例应用程序用于展示 AIR 功能。您可以按照以下说明访问和安装范例应用程序：

1 下载并运行 AIR 范例应用程序。编译后的应用程序及源代码都可用。

2 若要下载并运行范例应用程序，请单击范例应用程序的 “Install Now” 按钮。系统会提示您安装并运行该应用程序。

3 如果您选择下载范例应用程序并稍后运行，请选择下载链接。您可以随时通过以下方式运行 AIR 应用程序：

   - 在 Windows 中，双击桌面上的应用程序图标或从 Windows 的 “ 开始 ” 菜单中选择该应用程序。
   - 在 Mac OS 中，双击应用程序图标，默认情况下该应用程序安装在用户目录的 Applications 文件夹中 （例如，在 Macintosh HD/Users/JoeUser/Applications/ 中）。

注：检查 AIR 发行说明以查看是否有这些说明的更新，发行说明位于以下位置：
http://www.adobe.com/go/learn_air_relnotes_cn。

# Adobe AIR 更新

Adobe 定期使用新功能或小问题修补程序更新 Adobe AIR。使用自动通知和更新功能，Adobe 可以在提供了 Adobe AIR 的更新版本时自动通知用户。

对 Adobe AIR 的更新可确保 Adobe AIR 正常工作，并且通常包含对安全性的重要更改。Adobe 建议用户，只要提供了新版本的 Adobe AIR，就更新到这个最新版本 （特别是在涉及安全性更新时）。

默认情况下，当启动 AIR 应用程序时，运行时检查更新是否可用。至少每两周执行一次更新检查。如果更新可用， AIR 在后台下载更新。

通过使用 AIR SettingsManager 应用程序，用户可以禁用自动更新功能。 http://airdownload.adobe.com/air/applications/SettingsManager/SettingsManager.air 上提供了可供下载的 AIR SettingsManager 应用程序。

Adobe AIR 的正常安装过程包括连接到 http://airinstall.adobe.com 以发送有关安装环境的基本信息，例如操作系统版本和语言。仅在每次安装完成后发出此信息且 Adobe 可通过此信息确认安装是否成功。未收集或传输任何个人可识别的信息。

#### 更新捕获运行时

如果使用捕获运行时捆绑分发应用程序，则捕获运行时不会自动更新。为了用户的安全，当发布相关安全更改时，必须监控 Adobe 发布的更新，并使用新的运行时版本更新应用程序。

# 第 3 章：使用 AIR API

Adobe® AIR® 包括不适用于 Adobe® Flash® Player 中运行的 SWF 内容的功能。

**ActionScript 3.0** 开发人员
Adobe AIR API 相关内容在下列两书中进行了介绍：

- ActionScript 3.0 开发人员指南
- 用于 Adobe Flash Platform 的 ActionScript 3.0 参考

**HTML** 开发人员
在构建基于 HTML 的 AIR 应用程序时，可用于 JavaScript 中的 API 通过 AIRAliases.js 文件获得（请参阅通过 JavaScript 访问 AIR API 类），有关此类 API 的相关内容在下列两书中进行了介绍：

- Adobe AIR HTML 开发人员指南
- 针对 HTML 开发人员的 Adobe AIR API 参考

## 特定于 AIR 的 ActionScript 3.0 类

下表列出了特定于 Adobe AIR 的运行时类，它们不可用于在浏览器中的 Adobe® Flash® Player 中运行的 SWF 内容。

**HTML** 开发人员
通过 AIRAliases.js 文件获得、可用于 JavaScript 的类在针对 HTML 开发人员的 Adobe AIR API 参考中列出。

| 类 | ActionScript 3.0 包 | AIR 版本中添加了以下内容 |
|---|---|---|
| ARecord | flash.net.dns | 2.0 |
| AAAARecord | flash.net.dns | 2.0 |
| ApplicationUpdater | air.update | 1.5 |
| ApplicationUpdaterUI | air.update | 1.5 |
| AudioPlaybackMode | flash.media | 3.0 |
| AutoCapitalize | flash.text | 3.0 |
| BrowserInvokeEvent | flash.events | 1.0 |
| CameraPosition | flash.media | 3.0 |
| CameraRoll | flash.media | 2.0 |
| CameraRollBrowseOptions | flash.media | 3.0 |
| CameraUI | flash.media | 2.5 |
| CertificateStatus | flash.security | 2.0 |
| CompressionAlgorithm | flash.utils | 1.0 |

| 类 | ActionScript 3.0 包 | AIR 版本中添加了以下内容 |
| --- | --- | --- |
| DatagramSocket | flash.net | 2.0 |
| DatagramSocketDataEvent | flash.events | 2.0 |
| DNSResolver | flash.net.dns | 2.0 |
| DNSResolverEvent | flash.events | 2.0 |
| DockIcon | flash.desktop | 1.0 |
| DownloadErrorEvent | air.update.events | 1.5 |
| DRMAuthenticateEvent | flash.events | 1.0 |
| DRMDeviceGroup | flash.net.drm | 3.0 |
| DRMDeviceGroupErrorEvent | flash.net.drm | 3.0 |
| DRMDeviceGroupEvent | flash.net.drm | 3.0 |
| DRMManagerError | flash.errors | 1.5 |
| EncryptedLocalStore | flash.data | 1.0 |
| ExtensionContext | flash.external | 2.5 |
| File | flash.filesystem | 1.0 |
| FileListEvent | flash.events | 1.0 |
| FileMode | flash.filesystem | 1.0 |
| FileStream | flash.filesystem | 1.0 |
| FocusDirection | flash.display | 1.0 |
| GameInput | flash.ui | 3.0 |
| GameInputControl | flash.ui | 3.0 |
| GameInputControlType | flash.ui | 3.0 |
| GameInputDevice | flash.ui | 3.0 |
| GameInputEvent | flash.ui | 3.0 |
| GameInputFinger | flash.ui | 3.0 |
| GameInputHand | flash.ui | 3.0 |
| Geolocation | flash.sensors | 2.0 |
| GeolocationEvent | flash.events | 2.0 |
| HTMLHistoryItem | flash.html | 1.0 |
| HTMLHost | flash.html | 1.0 |
| HTMLLoader | flash.html | 1.0 |
| HTMLPDFCapability | flash.html | 1.0 |
| HTMLSWFCapabiltiy | flash.html | 2.0 |
| HTMLUncaughtScriptExceptionEvent | flash.events | 1.0 |
| HTMLWindowCreateOptions | flash.html | 1.0 |

| 类 | ActionScript 3.0 包 | AIR 版本中添加了以下内容 |
| --- | --- | --- |
| Icon | flash.desktop | 1.0 |
| IFilePromise | flash.desktop | 2.0 |
| ImageDecodingPolicy | flash.system | 2.6 |
| InteractiveIcon | flash.desktop | 1.0 |
| InterfaceAddress | flash.net | 2.0 |
| InvokeEvent | flash.events | 1.0 |
| InvokeEventReason | flash.desktop | 1.5.1 |
| IPVersion | flash.net | 2.0 |
| IURIDereferencer | flash.security | 1.0 |
| LocationChangeEvent | flash.events | 2.5 |
| MediaEvent | flash.events | 2.5 |
| MediaPromise | flash.media | 2.5 |
| MediaType | flash.media | 2.5 |
| MXRecord | flash.net.dns | 2.0 |
| NativeApplication | flash.desktop | 1.0 |
| NativeDragActions | flash.desktop | 1.0 |
| NativeDragEvent | flash.events | 1.0 |
| NativeDragManager | flash.desktop | 1.0 |
| NativeDragOptions | flash.desktop | 1.0 |
| NativeMenu | flash.display | 1.0 |
| NativeMenuItem | flash.display | 1.0 |
| NativeProcess | flash.desktop | 2.0 |
| NativeProcessExitEvent | flash.events | 2.0 |
| NativeProcessStartupInfo | flash.desktop | 2.0 |
| NativeWindow | flash.display | 1.0 |
| NativeWindowBoundsEvent | flash.events | 1.0 |
| NativeWindowDisplayState | flash.display | 1.0 |
| NativeWindowDisplayStateEvent | flash.events | 1.0 |
| NativeWindowInitOptions | flash.display | 1.0 |
| NativeWindowRenderMode | flash.display | 3.0 |
| NativeWindowResize | flash.display | 1.0 |
| NativeWindowSystemChrome | flash.display | 1.0 |
| NativeWindowType | flash.display | 1.0 |
| NetworkInfo | flash.net | 2.0 |

| 类 | ActionScript 3.0 包 | AIR 版本中添加了以下内容 |
| --- | --- | --- |
| NetworkInterface | flash.net | 2.0 |
| NotificationType | flash.desktop | 1.0 |
| OutputProgressEvent | flash.events | 1.0 |
| PaperSize | flash.printing | 2.0 |
| PrintMethod | flash.printing | 2.0 |
| PrintUIOptions | flash.printing | 2.0 |
| PTRRecord | flash.net.dns | 2.0 |
| ReferencesValidationSetting | flash.security | 1.0 |
| ResourceRecord | flash.net.dns | 2.0 |
| RevocationCheckSettings | flash.security | 1.0 |
| Screen | flash.display | 1.0 |
| ScreenMouseEvent | flash.events | 1.0 |
| SecureSocket | flash.net | 2.0 |
| SecureSocketMonitor | air.net | 2.0 |
| ServerSocket | flash.net | 2.0 |
| ServerSocketConnectEvent | flash.events | 2.0 |
| ServiceMonitor | air.net | 1.0 |
| SignatureStatus | flash.security | 1.0 |
| SignerTrustSettings | flash.security | 1.0 |
| SocketMonitor | air.net | 1.0 |
| SoftKeyboardType | flash.text | 3.0 |
| SQLCollationType | flash.data | 1.0 |
| SQLColumnNameStyle | flash.data | 1.0 |
| SQLColumnSchema | flash.data | 1.0 |
| SQLConnection | flash.data | 1.0 |
| SQLError | flash.errors | 1.0 |
| SQLErrorEvent | flash.events | 1.0 |
| SQLErrorOperation | flash.errors | 1.0 |
| SQLEvent | flash.events | 1.0 |
| SQLIndexSchema | flash.data | 1.0 |
| SQLMode | flash.data | 1.0 |
| SQLResult | flash.data | 1.0 |
| SQLSchema | flash.data | 1.0 |
| SQLSchemaResult | flash.data | 1.0 |

| 类 | ActionScript 3.0 包 | AIR 版本中添加了以下内容 |
|---|---|---|
| SQLStatement | flash.data | 1.0 |
| SQLTableSchema | flash.data | 1.0 |
| SQLTransactionLockType | flash.data | 1.0 |
| SQLTriggerSchema | flash.data | 1.0 |
| SQLUpdateEvent | flash.events | 1.0 |
| SQLViewSchema | flash.data | 1.0 |
| SRVRecord | flash.net.dns | 2.0 |
| StageAspectRatio | flash.display | 2.0 |
| StageOrientation | flash.display | 2.0 |
| StageOrientationEvent | flash.events | 2.0 |
| StageText | flash.text | 3.0 |
| StageTextInitOptions | flash.text | 3.0 |
| StageWebView | flash.media | 2.5 |
| StatusFileUpdateErrorEvent | air.update.events | 1.5 |
| StatusFileUpdateEvent | air.update.events | 1.5 |
| StatusUpdateErrorEvent | air.update.events | 1.5 |
| StatusUpdateEvent | air.update.events | 1.5 |
| StorageVolume | flash.filesystem | 2.0 |
| StorageVolumeChangeEvent | flash.events | 2.0 |
| StorageVolumeInfo | flash.filesystem | 2.0 |
| SystemIdleMode | flash.desktop | 2.0 |
| SystemTrayIcon | flash.desktop | 1.0 |
| TouchEventIntent | flash.events | 3.0 |
| UpdateEvent | air.update.events | 1.5 |
| Updater | flash.desktop | 1.0 |
| URLFilePromise | air.desktop | 2.0 |
| URLMonitor | air.net | 1.0 |
| URLRequestDefaults | flash.net | 1.0 |
| XMLSignatureValidator | flash.security | 1.0 |

# 具有特定于 AIR 功能的 Flash Player 类

以下类可用于在浏览器中运行的 SWF 内容，但 AIR 提供了附加的属性或方法：

| 包 | 类 | 属性、方法或事件 | AIR 版本中添加了以下内容 |
| --- | --- | --- | --- |
| flash.desktop | Clipboard | supportsFilePromise | 2.0 |
| | ClipboardFormats | BITMAP_FORMAT | 1.0 |
| | | FILE_LIST_FORMAT | 1.0 |
| | | FILE_PROMISE_LIST_FORMAT | 2.0 |
| | | URL_FORMAT | 1.0 |
| flash.display | LoaderInfo | childSandboxBridge | 1.0 |
| | | parentSandboxBridge | 1.0 |
| | Stage | assignFocus() | 1.0 |
| | | autoOrients | 2.0 |
| | | deviceOrientation | 2.0 |
| | | nativeWindow | 1.0 |
| | | orientation | 2.0 |
| | | orientationChange 事件 | 2.0 |
| | | orientationChanging 事件 | 2.0 |
| | | setAspectRatio | 2.0 |
| | | setOrientation | 2.0 |
| | | softKeyboardRect | 2.6 |
| | | supportedOrientations | 2.6 |
| | | supportsOrientationChange | 2.0 |
| | NativeWindow | owner | 2.6 |
| | | listOwnedWindows | 2.6 |
| | NativeWindowInitOptions | owner | 2.6 |

| 包 | 类 | 属性、方法或事件 | AIR 版本中添加了以下内容 |
| --- | --- | --- | --- |
| flash.events | Event | CLOSING | 1.0 |
| | | DISPLAYING | 1.0 |
| | | PREPARING | 2.6 |
| | | EXITING | 1.0 |
| | | HTML_BOUNDS_CHANGE | 1.0 |
| | | HTML_DOM_INITIALIZE | 1.0 |
| | | HTML_RENDER | 1.0 |
| | | LOCATION_CHANGE | 1.0 |
| | | NETWORK_CHANGE | 1.0 |
| | | STANDARD_ERROR_CLOSE | 2.0 |
| | | STANDARD_INPUT_CLOSE | 2.0 |
| | | STANDARD_OUTPUT_CLOSE | 2.0 |
| | | USER_IDLE | 1.0 |
| | | USER_PRESENT | 1.0 |
| | HTTPStatusEvent | HTTP_RESPONSE_STATUS | 1.0 |
| | | responseHeaders | 1.0 |
| | | responseURL | 1.0 |
| | KeyboardEvent | commandKey | 1.0 |
| | | controlKey | 1.0 |

| 包 | 类 | 属性、方法或事件 | AIR 版本中添加了以下内容 |
| --- | --- | --- | --- |
| flash.net | FileReference | extension | 1.0 |
| | | httpResponseStatus 事件 | 1.0 |
| | | uploadUnencoded() | 1.0 |
| | NetStream | drmAuthenticate 事件 | 1.0 |
| | | onDRMContentData 事件 | 1.5 |
| | | preloadEmbeddedData() | 1.5 |
| | | resetDRMVouchers() | 1.0 |
| | | setDRMAuthenticationCredentials() | 1.0 |
| | URLRequest | authenticate | 1.0 |
| | | cacheResponse | 1.0 |
| | | followRedirects | 1.0 |
| | | idleTimeout | 2.0 |
| | | manageCookies | 1.0 |
| | | useCache | 1.0 |
| | | userAgent | 1.0 |
| | URLStream | httpResponseStatus event | 1.0 |

| 包 | 类 | 属性、方法或事件 | AIR 版本中添加了以下内容 |
| --- | --- | --- | --- |
| flash.printing | PrintJob | active | 2.0 |
| | | copies | 2.0 |
| | | firstPage | 2.0 |
| | | isColor | 2.0 |
| | | jobName | 2.0 |
| | | lastPage | 2.0 |
| | | maxPixelsPerInch | 2.0 |
| | | paperArea | 2.0 |
| | | printableArea | 2.0 |
| | | printer | 2.0 |
| | | printers | 2.0 |
| | | selectPaperSize() | 2.0 |
| | | showPageSetupDialog() | 2.0 |
| | | start2() | 2.0 |
| | | supportsPageSetupDialog | 2.0 |
| | | terminate() | 2.0 |
| | PrintJobOptions | pixelsPerInch | 2.0 |
| | | printMethod | 2.0 |
| flash.system | Capabilities | languages | 1.1 |
| | LoaderContext | allowLoadBytesCodeExecution | 1.0 |
| | Security | APPLICATION | 1.0 |
| flash.ui | KeyLocation | D_PAD | 2.5 |

以上多数新增属性和方法都只对 AIR 应用程序安全沙箱中的内容可用。但 URLRequest 类中的新增成员对于其他沙箱中运行的内容也可用。

ByteArray.compress() 和 ByteArray.uncompress() 方法都包含一个新的 algorithm 参数，供您在 deflate 和 zlib 压缩之间选择。此参数仅对 AIR 中运行的内容可用。

# 特定于 AIR 的 Flex 组件

开发 Adobe AIR 内容时可以使用以下 Adobe® Flex™ MX 组件：

- FileEvent
- FileSystemComboBox
- FileSystemDataGrid
- FileSystemEnumerationMode

- FileSystemHistoryButton
- FileSystemList
- FileSystemSizeDisplayMode
- FileSystemTree
- FlexNativeMenu
- HTML
- Window
- WindowedApplication
- WindowedSystemManager

另外， Flex 4 包括下列 spark AIR 组件：

- Window
- WindowedApplication

有关 AIR Flex 组件的详细信息，请参阅使用 Flex AIR 组件。

# 第 4 章：适用于 AIR 开发的 Adobe Flash Platform 工具

您可以使用下列 Adobe Flash Platform 开发工具开发 AIR 应用程序。

对于 ActionScript 3.0 （Flash 和 Flex）开发人员：

- Adobe Flash Professional （请参阅为 AIR 发布）
- Adobe Flex 3.x 和 4.x SDK （请参阅第 17 页的 " 安装 Flex SDK" 和第 125 页的 "AIR Developer Tool (ADT)"）
- Adobe Flash Builder （请参阅使用 Flash Builder 开发 AIR 应用程序）

对于 HTML 和 Ajax 开发人员：

- Adobe AIR SDK （请参阅第 16 页的 " 安装 AIR SDK" 和第 125 页的 "AIR Developer Tool (ADT)"）
- Adobe Dreamweaver CS3、 CS4、 CS5 （请参阅 Dreamweaver 的 AIR 扩展）

## 安装 AIR SDK

Adobe AIR SDK 中包含以下命令行工具，可用于启动和打包应用程序：

**AIR Debug Launcher (ADL)** 允许您在不进行安装的情况下运行 AIR 应用程序。请参阅 第 121 页的 "AIR Debug Launcher (ADL)"。

**AIR Development Tool (ADT)** 将 AIR 应用程序打包为可分发的安装包。请参阅第 125 页的 "AIR Developer Tool (ADT)"。

AIR 命令行工具要求必须在计算机上安装 Java。可以使用 JRE 或 JDK （1.5 或更高版本）中的 Java 虚拟机。可以在 /http://java.sun.com/ 上找到 Java JRE 和 Java JDK。

要运行 ADT 工具，至少需要 2GB 的计算机内存。

注：最终用户运行 AIR 应用程序时不需要 Java。

有关使用 AIR SDK 构建 AIR 应用程序的简介，请参阅第 30 页的 " 使用 AIR SDK 创建第一个基于 HTML 的 AIR 应用程序 "。

### 下载并安装 AIR SDK

可以按照以下说明下载并安装 AIR SDK：

#### 在 **Windows** 中安装 **AIR SDK**

- 下载 AIR SDK 安装文件。
- AIR SDK 按标准归档文件进行分发。若要安装 AIR，请将 SDK 的内容提取到计算机上的一个文件夹 （例如：C:\Program Files\Adobe\AIRSDK 或 C:\AIRSDK）中。
- ADL 和 ADT 工具包含在 AIR SDK 的 bin 文件夹中；请将此文件夹的路径添加到 PATH 环境变量中。

#### 在 **Mac OS X** 中安装 **AIR SDK**

- 下载 AIR SDK 安装文件。

- AIR SDK 按标准归档文件进行分发。若要安装 AIR，请将 SDK 的内容提取到计算机上的一个文件夹 （例如：/Users/<userName>/Applications/AIRSDK）中。
- ADL 和 ADT 工具包含在 AIR SDK 的 bin 文件夹中；请将此文件夹的路径添加到 PATH 环境变量中。

在 **Linux** 中安装 **AIR SDK**

- 此 SDK 是以 tbz2 格式提供的。
- 要安装此 SDK，请创建一个要在其中解压缩此 SDK 的文件夹，然后使用以下命令：tar -jxvf <path to AIR-SDK.tbz2>

有关使用 AIR SDK 工具快速入门的信息，请参阅 " 使用命令行工具创建 AIR 应用程序 "。

## AIR SDK 中的内容

下表介绍了 AIR SDK 中所包含文件的用途：

| SDK 文件夹 | 文件 / 工具描述 |
|---|---|
| bin | 使用 AIR Debug Launcher (ADL)，可以在不先打包和安装 AIR 应用程序的情况下运行它。有关使用此工具的信息，请参阅 第 121 页的 "AIR Debug Launcher (ADL)"。<br><br>AIR Developer Tool (ADT) 会将应用程序打包为 AIR 文件以便分发。有关使用此工具的信息，请参阅第 125 页的 "AIR Developer Tool (ADT)"。 |
| frameworks | libs 目录包含可在 AIR 应用程序中使用的代码库。<br><br>projects 目录包含用于编译的 SWF 和 SWC 库的代码。 |
| include | include 目录包含用于编写本机扩展的 C 语言标头文件。 |
| install | install 目录包含 Android 设备的 Windows USB 启动程序。（这些驱动程序由 Google 提供，包含在 Android SDK 中。） |
| lib | 包含 AIR SDK 工具的支持代码。 |
| runtimes | 适用于桌面和适用于移动设备的 AIR 运行时。<br><br>桌面运行时由 ADL 用以在打包或安装 AIR 应用程序之前启动此类应用程序。<br><br>适用于 Android 的 AIR 运行时 （APK 包）可安装在 Android 设备或模拟器上，用于开发和测试。设备和模拟器使用单独的 APK 包。（从 Android Market 可以获取适用于 Android 的公共 AIR 运行时。） |
| samples | 此文件夹包含示例应用程序描述符文件、无缝安装功能示例 (badge.swf) 以及默认的 AIR 应用程序图标。 |
| templates | descriptor-template.xml - 应用程序描述符文件的模板，每个 AIR 应用程序都需要该模板。有关应用程序描述符文件的详细描述，请参阅第 154 页的 "AIR 应用程序描述符文件 "。<br><br>在此文件夹中，还可找到 AIR 每个发行版的应用程序描述符 XML 结构的架构文件。 |

# 安装 Flex SDK

要使用 Adobe® Flex™ 开发 Adobe® AIR® 应用程序，可以选择下列方式：

- 您可以下载并安装 Adobe® Flash® Builder™，其中提供了创建 Adobe AIR 项目以及测试、调试和打包 AIR 应用程序的集成工具。请参阅第 19 页的 " 在 Flash Builder 中创建第一个桌面 Flex AIR 应用程序 "。
- 您可以下载 Adobe® Flex™ SDK 并使用您最喜爱的文本编辑器和命令行工具开发 Flex AIR 应用程序。

有关使用 Flex SDK 构建 AIR 应用程序的简介，请参阅第 33 页的 “ 使用 Flex SDK 创建第一个桌面 AIR 应用程序 ”。

## 安装 Flex SDK

使用命令行工具生成 AIR 应用程序要求计算机上必须已安装 Java。可以使用 JRE 或 JDK （1.5 或更高版本）中的 Java 虚拟机。 http://java.sun.com/ 上提供了 Java JRE 和 JDK。

注：最终用户运行 AIR 应用程序时不需要 Java。

Flex SDK 提供了相关的 AIR API 和命令行工具，可用于对 AIR 应用程序进行打包、编译和调试。

1 如果尚未下载 Flex SDK，请从以下网址下载：http://opensource.adobe.com/wiki/display/flexsdk/Downloads。

2 将 SDK 的内容放到一个文件夹 （例如 Flex SDK）中。

3 复制 AIR SDK 的内容并覆盖 Flex SDK 中的文件。

   注：在 Mac 计算机上，确保您复制或替换的是 SDK 文件夹中的个别文件，而不是整个目录。默认情况下，将 Mac 上的某个目录复制到同名目录上，会删除目标目录中的现有文件，而不会合并两个目录的内容。可在终端窗口中使用 ditto 命令来将 AIR SDK 合并到 Flex SDK 中：ditto air_sdk_folder flex_sdk_folder

4 命令行 AIR 实用程序位于 bin 文件夹中。

# 设置外部 SDK

为 Android 和 iOS 开发应用程序要求您从平台制造商处下载供给文件、 SDK 或其他开发工具。

有关下载和安装 Android SDK 的信息，请参阅 Android 开发人员：安装 SDK。从 AIR 2.6 开始，不需要下载 Android SDK。 AIR SDK 现已包含安装和启动 APK 包所需的基本组件。不过，对于各种开发任务 （包括创建和运行软件仿真器以及获取设备屏幕截图）来说， Android SDK 仍然非常有用。

iOS 开发不需要外部 SDK。但是，需要专用证书和供给配置文件。有关更多信息，请参阅从 Apple 获取开发人员文件。

# 第 5 章：创建第一个 AIR 应用程序

## 在 Flash Builder 中创建第一个桌面 Flex AIR 应用程序

为了通过实际操作较快地说明 Adobe® AIR® 的工作原理，请按照以下说明使用 Adobe® Flash® Builder 创建并打包一个简单的基于 SWF 文件的 AIR"Hello World" 应用程序。

如果尚未执行这些操作，请下载并安装 Flash Builder。此外，还需要下载并安装最新版本的 Adobe AIR，位于此处：www.adobe.com/go/air_cn。

### 创建 AIR 项目

Flash Builder 包括用于开发和打包 AIR 应用程序的工具。

通过定义一个新项目来开始在 Flash Builder 和 Flex Builder 中创建 AIR 应用程序，其方式与创建其他基于 Flex 的应用程序项目的方式相同。

1 打开 Flash Builder。
2 依次选择 " 文件 ">" 新建 ">"Flex 项目 "。
3 输入项目名称，例如 AIRHelloWorld。
4 在 Flex 中， AIR 应用程序被视为一种应用程序类型。有两种类型可供您选择：
   - 在 Adobe® Flash® Player 中运行的 Web 应用程序
   - 在 Adobe AIR 中运行的桌面应用程序

   选择 Desktop 作为应用程序类型。
5 单击 " 完成 " 以创建此项目。

AIR 项目最初由两个文件组成：主 MXML 文件和应用程序 XML 文件 （称为应用程序描述符文件）。后一个文件用于指定应用程序的属性。

有关详细信息，请参阅使用 Flash Builder 开发 AIR 应用程序。

### 编写 AIR 应用程序代码

若要编写 "Hello World" 应用程序代码，请编辑在编辑器中打开的应用程序 MXML 文件 (AIRHelloWorld.mxml)。（如果该文件尚未打开，请使用 Project Navigator 打开该文件。）

桌面上的 Flex AIR 应用程序内含在 MXML WindowedApplication 标签内。MXML WindowedApplication 标签用于创建一个简单窗口，此窗口包括诸如标题栏和关闭按钮之类的基本窗口控件。

1 对 WindowedApplication 组件添加 title 属性，并为此属性赋予值 "Hello World"：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                       title="Hello World">
</s:WindowedApplication>
```

2 对此应用程序添加 Label 组件 （将其置于 WindowedApplication 标签内），将 Label 组件的 text 属性设置为 "Hello AIR"，并设置布局限制以使其保持居中，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                       title="Hello World">

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

**3** 在刚输入的 Label 组件标签之前，紧跟 WindowedApplication 开始标签添加以下样式块。

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    s|WindowedApplication
    {

skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
        background-color:#999999;
        background-alpha:"0.7";
    }
</fx:Style>
```

这些样式设置将应用于整个应用程序，用稍稍透明的灰色呈现窗口。

现在，应用程序代码类似于如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                       title="Hello World">

    <fx:Style>
        @namespace s "library://ns.adobe.com/flex/spark";
        s|WindowedApplication
        {

skinClass:ClassReference("spark.skins.spark.SparkChromeWindowedApplicationSkin");
            background-color:#999999;
            background-alpha:"0.7";
        }
    </fx:Style>

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

接下来，您要更改应用程序描述符中的一些设置，以允许应用程序变得透明：

**1** 在 Flex Navigator 窗格中，在项目的源目录中找到该应用程序的描述符文件。如果项目名为 AIRHelloWorld，则该文件应该名为 AIRHelloWorld-app.xml。

**2** 双击该应用程序描述符文件，以便在 Flash Builder 中进行编辑。

**3** 在 XML 代码中，找到 systemChrome 和 transparent 属性 （属于 initialWindow 属性）的注释行。删除注释。（删除 "<!--" 和 "-->" 注释分隔符。）

**4** 将 systemChrome 属性的文本值设为 none，如下所示：

```
<systemChrome>none</systemChrome>
```

**5** 将 transparent 属性的文本值设为 true，如下所示：

```
<transparent>true</transparent>
```

**6** 保存该文件。

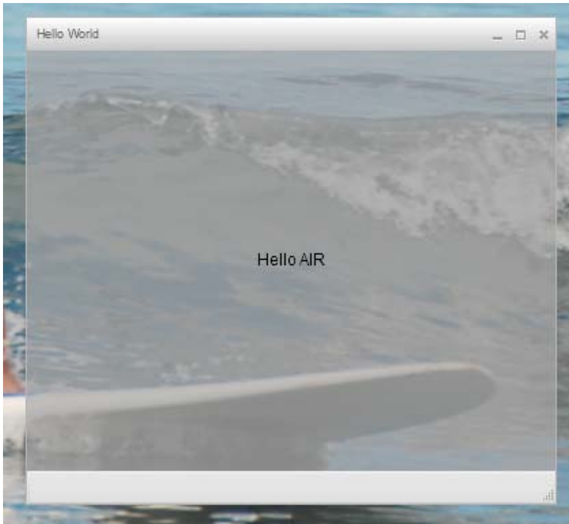
## 测试 AIR 应用程序

若要测试编写完的应用程序代码，请在调试模式下运行它。

1 单击主工具栏上的“调试”按钮 。

也可以选择“运行”>“调试”>“AIRHelloWorld 命令”。

生成的 AIR 应用程序应如下例所示：




2 通过使用 Label 控件的 horizontalCenter 和 verticalCenter 属性，文本位于窗口的中心位置。移动窗口或调整窗口大小，像对任何其他桌面应用程序所做的那样。

注：如果此应用程序不编译，请修正输入代码时无意间造成的语法错误或拼写错误。错误和警告显示在 Flash Builder 的“问题”视图中。

## 打包 AIR 应用程序、为应用程序签名和运行应用程序

现在已准备好将“Hello World”应用程序打包为 AIR 文件以进行分发。 AIR 文件是一个包含应用程序文件的归档文件，这里的应用程序文件为项目 bin 文件夹中包含的所有文件。在此简单示例中，这些文件为 SWF 和应用程序 XML 文件。将此 AIR 包分发给用户，用户随后使用此包安装此应用程序。此过程中的一个必需步骤是对包进行数字签名。

1 确保应用程序没有编译错误并按预期运行。

2 选择“项目”>“导出发行版”。

3 查看 AIRHelloWorld 项目和 AIRHelloWorld.mxml 应用程序是否已列出。

4 选择“导出为签名的 AIR 包”选项。然后单击“下一步”。

5 如果有现成的数字证书，请单击“浏览”找到并选中它。

6 如果必须创建新的自签名数字证书，请选择“创建”。

7 输入所需信息，然后单击“确定”。

8 单击“完成”以生成名为 AIRHelloWorld.air 的 AIR 包。

现在，既可以从 Flash Builder 的 Project Navigator 中安装和运行该应用程序，也可以通过双击该 AIR 文件从文件系统安装和运行该应用程序。

# 使用 Flash Professional 创建第一个桌面 AIR 应用程序

为了通过实际操作快速演示 Adobe® AIR® 的工作原理，请遵循本主题中的说明使用 Adobe® Flash® Professional 创建并打包一个简单的 “Hello World”AIR 应用程序。

如果尚未执行这些操作，请下载并安装 Adobe AIR，其位置为：www.adobe.com/go/air_cn。

## 在 Flash 中创建 Hello World 应用程序

在 Flash 中创建 Adobe AIR 应用程序与创建任何其他 FLA 文件非常相似。以下步骤将指导您完成使用 Flash Professional 创建简单的 Hello World 应用程序的过程。

创建 **Hello World** 应用程序

1 启动 Flash。
2 在 " 欢迎 " 屏幕上，单击 “AIR” 以创建具有 Adobe AIR 发布设置的空 FLA 文件。
3 在 " 工具 " 面板中选择 " 文本 " 工具并在舞台的中央创建静态文本字段 ( 默认值 ) 。将其宽度设置为足以包含 15-20 个字符。
4 在该文本字段中输入文本 “Hello World”。
5 保存该文件，为其指定一个名称 （例如， HelloAIR）。

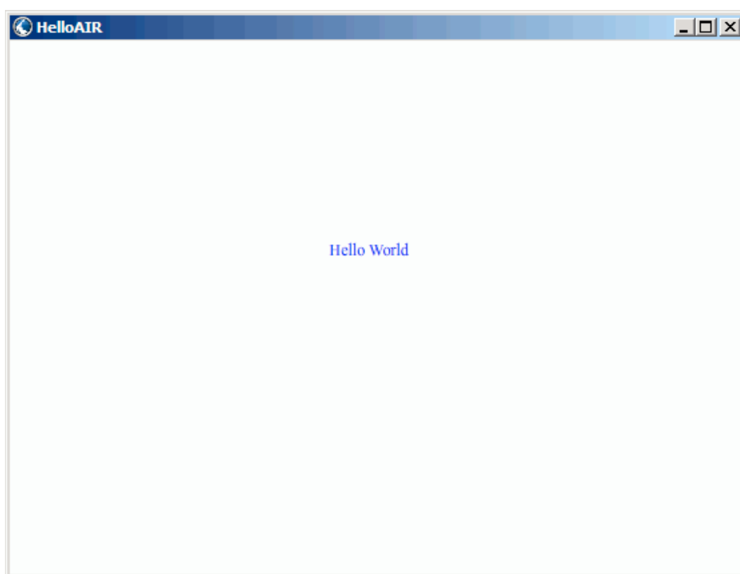
## 测试应用程序

1 按 Ctrl + Enter 或选择 " 控制 "->" 测试影片 "->" 测试 " 以在 Adobe AIR 中测试应用程序。
2 若要使用 " 调试影片 " 功能，请首先向该应用程序添加 ActionScript 代码。可以通过添加类似以下的 trace 语句来快速尝试该操作：

```
trace("Running AIR application using Debug Movie");
```

3 按 Ctrl + Shift + Enter 或选择 " 调试 "->" 调试影片 "->" 调试 " 以对该应用程序运行 " 调试影片 "。

Hello World 应用程序的外观如下图所示：



### 打包应用程序

1 选择 “ 文件 ”>“ 发布 ”。

2 使用现有数字证书对 Adobe AIR 包签名或按照下列步骤创建自签名证书:

   a 单击 “ 证书 ” 字段旁边的 “ 新建 ” 按钮。

   b 填写 “ 发布者名称 ”、 “ 部门 ”、 “ 组织名称 ”、 “ 电子邮件 ”、 “ 国家 / 地区 ”、 “ 密码 ” 和 “ 确认密码 ” 条目。

   c 指定证书的类型。“ 证书类型 ” 选项是指安全级别: 1024-RSA 使用 1024 位密钥 (不太安全), 2048-RSA 使用 2048 位密钥 (较为安全)。

   d 通过填写 “ 另存为 ” 条目或单击 “ 浏览 ...” 按钮浏览到文件夹位置，将信息保存到证书文件中。 (例如， C:/Temp/mycert.pfx)。完成后，请单击 “ 确定 ”。

   e Flash 将返回到 “ 数字签名 ” 对话框。您创建的自签名证书的路径和文件名均显示在 “ 证书 ” 文本框中。如果未显示，请输入路径和文件名或者单击 “ 浏览 ” 按钮找到并选择它。

   f 在“数字签名”对话框的“密码”文本字段中输入步骤 b 中所指定的同一个密码。有关对 Adobe AIR 应用程序进行签名的更多信息，请参阅第 143 页的 “ 对 AIR 文件进行数字签名 ”。

3 若要创建应用程序和安装程序文件，请单击 “ 发布 ” 按钮。(在 Flash CS4 和 CS5 中，请单击 “ 确定 ” 按钮。) 在创建 AIR 文件之前，必须执行 “ 测试影片 ” 或 “ 调试影片 ” 来创建 SWF 文件和 application.xml 文件。

4 若要安装应用程序，请双击保存该应用程序的同一个文件夹中的 AIR 文件 (application.air)。

5 在 “ 应用程序安装 ” 对话框中，单击 “ 安装 ” 按钮。

6 检查 “ 安装首选参数 ” 和 “ 位置 ” 设置，确保选中了 “ 安装后启动应用程序 ” 复选框。然后单击 “ 继续 ”。

7 在出现 “ 安装已完成 ” 消息时，单击 “ 完成 ”。

## 在 Flash Professional 中创建您的第一个 AIR for Android 应用程序

若要开发 AIR for Android 应用程序，必须从 Adobe Labs 下载 Flash Professional CS5 for Android 扩展。

同时还必须从 Android 网站下载并安装 Android SDK，如 Android 开发人员：安装 SDK 中所述。

创建项目

1 打开 Flash Professional CS5

2 创建新的 AIR for Android 项目。

   Flash Professional 主屏幕包括一个用于创建 AIR for Android 应用程序的链接。您也可以选择 “ 文件 ”>“ 新建 ”，然后选择 “AIR for Android” 模板。

3 将文档另存为 HelloWorld.fla

编写代码

由于此教程并不是关于编写代码的，因此只要使用 “ 文本 ” 工具在舞台上写入 “Hello, World!” 即可。

设置应用程序属性

1 选择 “ 文件 ”>“AIR Android 设置 ”。

2 在 “ 常规 ” 选项卡中，设置下列内容：

   • 输出文件：HelloWorld.apk

- 应用程序名称：HelloWorld
- 应用程序 ID：HelloWorld
- 版本号：0.0.1
- 高宽比：纵向

3 在 " 部署 " 选项卡上，设置下列内容：

- 证书：指向有效的 AIR 代码签名证书。可以单击 " 创建 " 按钮以创建新的证书。（通过 Android Marketplace 部署的 Android 应用程序必须具有有效期至少到 2033 年的证书。在 " 密码 " 字段中输入证书密码。
- Android 部署类型：调试
- 发布后：同时选择这两个选项
- 在 Android SDK 的工具子目录中输入到 ADB 工具的路径。

4 单击 " 确定 " 关闭 "Android 设置 " 对话框。

应用程序在此开发阶段不需要图标或权限。大多数 AIR for Android 应用程序都需要具有一定的权限才能访问受保护的功能。您仅应设置应用程序真正需要的那些权限，因为如果您的应用程序要求的权限太多，用户可能会拒绝该应用程序。

5 保存该文件。

在 **Android** 设备上安装应用程序并对其进行打包

1 确保在您的设备上已启用 USB 调试。您可以在 " 应用程序 ">" 开发 " 下的 Settings 应用程序中打开 USB 调试。

2 使用 USB 电缆将您的设备连接到计算机上。

3 如果您尚未安装 AIR 运行时，请通过转到 Android Market 并下载 Adobe AIR 进行安装。（也可以使用 第 133 页的 "ADT installRuntime 命令 " 本地安装 AIR。 AIR SDK 中包括在 Android 设备和仿真器上使用的 Android 包。）

4 选择 " 文件 ">" 发布 "。

Flash Professional 会创建 APK 文件，在连接的 Android 设备上安装应用程序，并启动该应用程序。

# 创建第一个用于 iOS 的 AIR 应用程序

**AIR 2.6** 或更高版本， **iOS 4.2** 或更高版本

只需使用 Adobe 工具即可编写和生成 iOS 应用程序并测试 iOS 应用程序的基本功能。但是，若要在设备上安装 iOS 应用程序并分发该应用程序，您必须加入 Apple iOS 开发人员计划（这是一项免费服务）。加入 iOS 开发人员计划之后，可以访问 iOS Provisioning Portal，通过该网站，可以从 Apple 获得在设备上安装应用程序所需的以下项目和文件，以供测试和后续分发之用。这些项目和文件包括：

- 开发和分发证书
- 应用程序 ID
- 开发和分发供给文件

## 创建应用程序内容

创建一个显示文本 "Hello world!" 的 SWF 文件可以使用 Flash Professional、Flash Builder 或其他 IDE 来执行此任务。此示例只使用了文本编辑器和 Flex SDK 中附带的命令行 SWF 编译器。

1 在适当的位置创建一个目录，用于存储您的应用程序文件。创建一个名为 HelloWorld.as 的文件，然后使用您喜欢的代码编辑器编辑该文件。

2 添加以下代码：

```
package{

    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.text.TextFieldAutoSize;

    public class HelloWorld extends Sprite
    {
        public function HelloWorld():void
        {
            var textField:TextField = new TextField();
            textField.text = "Hello World!";
            textField.autoSize = TextFieldAutoSize.LEFT;

            var format:TextFormat = new TextFormat();
            format.size = 48;

            textField.setTextFormat ( format );
            this.addChild( textField );
        }
    }
}
```

3 使用 amxmlc 编译器编译类：

```
amxmlc HelloWorld.as
```

将在相同文件夹中创建一个 SWF 文件：HelloWorld.swf。

注：此示例假设您已将环境路径变量设置为包含 amxmlc 的目录。有关设置路径的信息，请参阅第 237 页的 " 路径环境变量 "。此外，也可以键入 amxmlc 以及此示例中使用的其他命令行工具的完整路径。

## 创建应用程序的图标图片和初始屏幕图片

所有 iOS 应用程序都具有在 iTunes 应用程序的用户界面和设备屏幕上显示的图标。

1 在您的项目目录中创建一个目录并将其命名为图标。

2 在该图标目录中创建三个 PNG 文件。将它们分别命名为 Icon_29.png、 Icon_57.png 和 Icon_512.png。

3 编辑这些 PNG 文件为应用程序创建适当的图片。这些文件必须为 29×29 像素、 57×57 像素和 512×512 像素。对于此测试，您可以仅使用纯色正方形作为图片。

注：将应用程序提交给 Apple 应用程序库时，使用像素为 512 的 JPG 版本文件 （而不是 PNG 版本）。测试应用程序的开发版本时使用 PNG 版本。

在 iPhone 上加载应用程序时，所有 iPhone 应用程序均显示初始图像。您可以在 PNG 文件中定义初始图像：

1 在主开发目录中，创建名为 Default.png 的 PNG 文件。（不要将此文件放在图标子目录中。确保将此文件命名为 Default.png，使用大写 D。）

2 编辑该文件，使其宽 320 像素，高 480 像素。此刻，内容可能显示为纯白色矩形。（您稍后可以进行更改。）

有关这些图形的详细信息，请参阅第 71 页的 " 应用程序图标 "。

## 创建应用程序描述符文件

创建应用程序描述符文件，该文件指定了应用程序的基本属性。您可以使用 IDE （如 Flash Builder）或文本编辑器来完成此任务。

1 在包含 HelloWorld.as 的项目文件夹中，创建一个名为 HelloWorld-app.xml 的 XML 文件。使用您喜欢的 XML 编辑器编辑该文件。

2 添加以下 XML 代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/2.7" minimumPatchLevel="0">
    <id>change_to_your_id</id>
    <name>Hello World iOS</name>
    <versionNumber>0.0.1</versionNumber>
    <filename>HelloWorld</filename>
    <supportedProfiles>mobileDevice</supportedProfiles>
    <initialWindow>
        <content>HelloWorld.swf</content>
        <title>Hello World!</title>
    </initialWindow>
    <icon>
        <image29x29>icons/AIRApp_29.png</image29x29>
        <image57x57>icons/AIRApp_57.png</image57x57>
        <image512x512>icons/AIRApp_512.png</image512x512>
    </icon>
</application>
```

为了简单起见，此示例仅设置了少数几个可用的属性。

注：如果您使用的是 AIR 2 或更早版本，则必须使用 <version> 元素，而非 <versionNumber> 元素。

3 更改应用程序 ID 使其与 iOS Provisioning Portal 中指定的应用程序 ID 相匹配。(不要包含位于 ID 开头的随机捆绑种子部分。)

4 使用 ADL 测试应用程序：

```
adl HelloWorld-app.xml -screensize iPhone
```

ADL 将在桌面上打开一个窗口，其中显示以下文本：Hello World!，如果未显示该文本，请检查源代码和应用程序描述符中的错误。

## 编译 IPA 文件

现在您可以使用 ADT 来编译 IPA 安装程序文件：您必须具有 Apple 开发人员证书、私钥（采用 P12 文件格式）和 Apple 开发供给配置文件。

使用以下选项运行 ADT 实用程序，使用自己的值替换 keystore、 storepass 和 provisioning-profile 值：

```
adt -package -target ipa-debug
    -keystore iosPrivateKey.p12 -storetype pkcs12 -storepass qwerty12
    -provisioning-profile ios.mobileprovision
    HelloWorld.ipa
    HelloWorld-app.xml
    HelloWorld.swf icons Default.png
```

（使用一个命令行；在此示例中添加换行符只是为了便于阅读。）

ADT 在项目目录中生成 iOS 应用程序的安装程序文件 HelloWorld.ipa。编译 IPA 文件可能需要几分钟的时间。

## 在设备上安装应用程序

安装 iOS 应用程序以进行测试：

1 打开 iTunes 应用程序。

2 如果您尚未执行此操作，请将该应用程序的供给配置文件添加到 iTunes。在 iTunes 中，选择 " 文件 ">" 添加到资料库 "。然后选择供给配置文件 （其文件类型为 mobileprovision）。

  此刻，要在开发人员设备上测试该应用程序，请使用开发供给配置文件。

  稍后，将应用程序分发给 iTunes Store 时，请使用分发配置文件。要临时分发应用程序 （不通过 iTunes Store 分发给多个设备），请使用临时供给配置文件。

  有关供给配置文件的更多信息，请参阅 第 56 页的 "iOS 设置 "。

3 如果已安装相同版本的应用程序， iTunes 的某些版本不会替换该应用程序。在这种情况下，从您的设备和 iTunes 中的应用程序列表中删除该应用程序。

4 双击您的应用程序的 IPA 文件。此时，您的应用程序应显示在 iTunes 中的应用程序列表中。

5 将您的设备连接到计算机上的 USB 端口。

6 在 iTunes 中，检查 " 应用程序 " 选项卡中是否存在该设备，并确保在要安装的应用程序的列表中选中了该应用程序。

7 选择左侧 iTunes 应用程序列表中的设备。然后单击 " 同步 " 按钮。完成同步后， Hello World 应用程序会显示在您的 iPhone 上。

如果未安装新版本，请将其从您的设备以及 iTunes 中的应用程序列表中删除，然后重新执行此过程。这可能是因为目前安装的版本使用的是相同的应用程序 ID 和版本。

## 编辑初始屏幕图形

在编译应用程序之前，您创建了一个 Default.png 文件 （请参见第 25 页的 " 创建应用程序的图标图片和初始屏幕图片 "）。当加载应用程序时，此 PNG 文件充当启动图像。在 iPhone 上测试应用程序时，您可能已注意到启动时会出现空白屏幕。

您应更改此图像，使之与您的应用程序 （"Hello World!"）的启动屏幕相匹配：

1 打开设备上的应用程序。当第一个 "Hello World" 文本出现时，按住 " 主屏幕 " 按钮 （位于屏幕下方）。在按住 " 主屏幕 " 按钮的同时，按下 " 睡眠 / 唤醒 " 按钮 （位于 iPhone 的顶部）。这会捕获一张屏幕快照并将其发送到摄像头卷。

2 通过从 iPhoto 或其他照片传输应用程序传输照片将此图像传输到您的开发计算机。（在 Mac OS 上，您也可以使用图像捕捉应用程序。）

  还可以通过电子邮件将照片发送到您的开发计算机：

  - 打开 Photos 应用程序。
  - 打开摄像头卷。
  - 打开您捕获的屏幕快照图像。
  - 点击此图像，然后点击左下角的 " 转发 " （箭头）按钮。然后单击 " 用电子邮件发送照片 " 按钮并将图像发送给自己。

3 使用 PNG 版本的屏幕捕获图像替换 Default.png 文件 （位于您的开发目录中）。

4 重新编译该应用程序 （请参阅第 26 页的 " 编译 IPA 文件 "）并将其重新安装在您的设备中。

该应用程序现在使用的是其加载的新启动屏幕。

注：您可以为 Default.png 文件创建任何图片，只要尺寸正确 （320 × 480 像素）即可。但是，通常最好将 Default.png 图像与应用程序的初始状态相匹配。

# 使用 Dreamweaver 创建第一个基于 HTML 的 AIR 应用程序

为了通过实际操作较快地说明 Adobe® AIR® 的工作原理，请按照以下说明使用 Adobe® AIR® Extension for Dreamweaver® 创建并打包一个简单的基于 HTML 的 AIR"Hello World" 应用程序。

如果尚未执行这些操作，请下载并安装 Adobe AIR，其位置为：www.adobe.com/go/air_cn。

有关安装 Dreamweaver 的 Adobe AIR 扩展的说明，请参阅安装 Dreamweaver 的 Adobe AIR 扩展。

有关该扩展的概述 （包括系统要求），请参阅 Dreamweaver 的 AIR 扩展。

注：对于桌面和 extendedDesktop 配置文件，只能开发基于 HTML 的 AIR 应用程序。不支持移动配置文件和 tv 配置文件。

## 准备应用程序文件

Adobe AIR 应用程序必须在 Dreamweaver 站点中定义一个起始页及其所有相关页：

1 启动 Dreamweaver 并确保定义一个站点。

2 打开一个新 HTML 页，方法是选择"文件">"新建"，然后在"页面类型"列中选择 HTML，在"布局"列中选择"无"，最后单击 " 创建 "。

3 在此新页面中，键入 **Hello World!**

 此示例极其简单，但如果需要，也可以根据喜好设定文本样式，将其他内容添加到此页面中，将其他页面链接到此起始页，等等。

4 将此页面另存为 hello_world.html （" 文件 ">" 保存 "）。确保将此文件保存在 Dreamweaver 站点中。

有关 Dreamweaver 站点的详细信息，请参阅 Dreamweaver 帮助。

## 创建 Adobe AIR 应用程序

1 确保在 Dreamweaver 文档窗口中打开 hello_world.html 页。（有关创建此页的说明，请参阅上一部分。）

2 选择 " 网站 ">"Air 应用程序设置 "。

 "AIR 应用程序和设置 " 对话框中的大多数所需设置都自动为您填充。但是，您必须选择应用程序的起始内容 （或起始页）。

3 单击 " 起始内容 " 选项旁的 " 浏览 " 按钮，导航到 hello_world.html 页，然后选择此页。

4 在 " 数字签名 " 选项的旁边，单击 " 设置 " 按钮。

 数字签名用于保证自软件作者创建应用程序起应用程序的代码未经更改或未受损坏，所有 Adobe AIR 应用程序均需要数字签名。

5 在"数字签名"对话框中，选择"使用数字证书为 AIR 包签名"，然后单击"创建"按钮。（如果已具有对某一数字证书的访问权限，则可以单击 " 浏览 " 按钮选择此数字证书。）

6 完成 " 自签名数字证书 " 对话框中的所需字段。您将需要输入您的名称，输入密码并确认，以及输入数字证书文件的名称。Dreamweaver 将在站点根目录中保存此数字证书。

7 单击 " 确定 " 以返回 " 数字签名 " 对话框。

8 在 " 数字签名 " 对话框中，输入您为您的数字证书指定的密码并单击 " 确定 "。

完成后的 “AIR 应用程序和安装程序设置 ” 对话框可能类似于如下内容：

有关所有对话框选项及如何对其编辑的详细说明，请参阅在 Dreamweaver 中创建 AIR 应用程序。

**9** 单击 “ 创建 AIR 文件 ” 按钮。

Dreamweaver 将创建相应的 Adobe AIR 应用程序文件并将其保存在站点根文件夹中。 Dreamweaver 还将创建 application.xml 文件并将其保存在相同的位置中。此文件的作用像清单一样，定义了应用程序的各种属性。

## 在桌面上安装应用程序

既然已创建应用程序文件，那就可以在任何桌面上安装该应用程序。

**1** 将相应的 Adobe AIR 应用程序文件移出 Dreamweaver 站点，并移到您的桌面或其他桌面上。

此步骤为可选步骤。实际上，如果愿意，可以从 Dreamweaver 站点目录直接将新建应用程序安装到计算机上。

**2** 双击应用程序可执行文件 （.air 文件）以安装应用程序。

## 预览 Adobe AIR 应用程序

可以随时预览将归属于 AIR 应用程序的页面。即，不必打包应用程序即可查看其安装后的样子。

**1** 确保在 Dreamweaver 文档窗口中打开 hello_world.html 页。

**2** 在 " 文档 " 工具栏上，单击 " 在浏览器中预览 / 调试 " 按钮，然后选择 " 在 AIR 中预览 "。

也可以按 Ctrl+Shift+F12 (Windows) 或 Cmd+Shift+F12 (Macintosh)。

当您预览此页时，您看到的实际上就是当用户在桌面上安装应用程序之后他们将看到的应用程序起始页。

# 使用 AIR SDK 创建第一个基于 HTML 的 AIR 应用程序

为了通过实际操作较快地说明 Adobe® AIR® 的工作原理，请遵循以下说明创建并打包一个简单的基于 HTML 的 AIR"Hello World" 应用程序。

开始前，必须已安装运行时并设置了 AIR SDK。本教程将涉及使用 AIR Debug Launcher (ADL) 和 AIR Developer Tool (ADT)。ADL 和 ADT 是命令行实用工具程序，可在 AIR SDK 的 bin 目录中找到（请参阅第 16 页的 " 安装 AIR SDK"）。本教程假定您已经熟悉从命令行运行程序并了解了如何针对您的操作系统设置所需的路径环境变量。

注：如果您是一名 Adobe® Dreamweaver® 用户，请阅读第 28 页的 " 使用 Dreamweaver 创建第一个基于 HTML 的 AIR 应用程序 "。

注：对于桌面和 extendedDesktop 配置文件，只能开发基于 HTML 的 AIR 应用程序。不支持移动配置文件和 tv 配置文件。

## 创建项目文件

每个基于 HTML 的 AIR 项目必须包含以下两个文件：可指定应用程序元数据的应用程序描述符文件，以及顶级 HTML 页。除了这两个所需文件外，此项目还包含一个 JavaScript 代码文件 AIRAliases.js，该文件可为 AIR API 类定义好记的别名变量。

**1** 创建一个名为 HelloWorld 的目录以包含项目文件。

**2** 创建一个名为 HelloWorld-app.xml 的 XML 文件。

**3** 创建一个名为 HelloWorld.html 的 HTML 文件。

**4** 将 AIR SDK 的 frameworks 文件夹中的 AIRAliases.js 复制到项目目录中。

## 创建 AIR 应用程序描述符文件

开始构建 AIR 应用程序前，应创建一个具有以下结构的 XML 应用程序描述符文件：

```
<application xmlns="...">
    <id>…</id>
    <versionNumber>…</versionNumber>
    <filename>…</filename>
    <initialWindow>
        <content>…</content>
        <visible>…</visible>
        <width>…</width>
        <height>…</height>
    </initialWindow>
</application>
```

**1** 打开要编辑的 HelloWorld-app.xml。

**2** 添加根 <application> 元素，包括 AIR 命名空间属性：

**<application xmlns="http://ns.adobe.com/air/application/2.7">** 命名空间的最后一段 "2.7" 指定应用程序所需的运行时版本。

**3** 添加 <id> 元素：

**<id>examples.html.HelloWorld</id>** 应用程序 ID 与发行商 ID （由 AIR 派生自用于对应用程序包进行签名的证书）一起唯一地标识了应用程序。应用程序 ID 用于安装、访问专用应用程序文件系统存储目录、访问专用加密存储以及应用程序间的通信。

4 添加 <versionNumber> 元素：

**<versionNumber>0.1</versionNumber>** 可帮助用户确定安装哪个版本的应用程序。

注：如果您使用的是 AIR 2 或更早版本，则必须使用 <version> 元素，而非 <versionNumber> 元素。

5 添加 <filename> 元素：

**<filename>HelloWorld</filename>** 用于操作系统中应用程序可执行文件、安装目录和对应用程序的其他引用的名称。

6 添加包含下列子元素的 <initialWindow> 元素，从而为初始应用程序窗口指定属性：

**<content>HelloWorld.html</content>** 标识 AIR 要加载的根 HTML 文件。

**<visible>true</visible>** 使窗口立即可见。

**<width>400</width>** 设置窗口宽度 （以像素为单位）。

**<height>200</height>** 设置窗口高度。

7 保存该文件。完整的应用程序描述符文件应如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
    <id>examples.html.HelloWorld</id>
    <versionNumber>0.1</versionNumber>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>HelloWorld.html</content>
        <visible>true</visible>
        <width>400</width>
        <height>200</height>
    </initialWindow>
</application>
```

此示例仅设置了几个可能的应用程序属性。有关应用程序属性的完整设置 （利用这些设置可以指定窗口镶边、窗口大小、透明度、默认安装目录、关联文件类型以及应用程序图标等），请参阅 第 154 页的 “AIR 应用程序描述符文件 ”。

## 创建应用程序 HTML 页

现在需要创建一个简单的 HTML 页以用作 AIR 应用程序的主文件。

1 打开要编辑的 HelloWorld.html 文件。添加以下 HTML 代码：

```
<html>
<head>
    <title>Hello World</title>
</head>
<body onLoad="appLoad()">
    <h1>Hello World</h1>
</body>
</html>
```

2 在该 HTML 的 <head> 部分，导入 AIRAliases.js 文件：

```
<script src="AIRAliases.js" type="text/javascript"></script>
```

AIR 针对 HTML window 对象定义一个名为 runtime 的属性。该 runtime 属性使用类的完全限定的包名称来提供对内置 AIR 类的访问。例如，若要创建 AIR File 对象，您可以在 JavaScript 中添加以下语句：

```
var textFile = new runtime.flash.filesystem.File("app:/textfile.txt");
```

AIRAliases.js 文件为最常用的 AIR API 定义了好记的别名。使用 AIRAliases.js，可以将对 File 类的引用缩短为以下形式：

```
var textFile = new air.File("app:/textfile.txt");
```

3 在 AIRAliases 脚本标签下方，添加另一个包含 JavaScript 函数的脚本标签来处理 onLoad 事件：

```
<script type="text/javascript">
function appLoad(){
    air.trace("Hello World");
}
</script>
```

appLoad() 函数仅调用 air.trace() 函数。使用 ADL 运行应用程序时，跟踪消息会输出到命令控制台。Trace 语句对于调试非常有用。

4 保存该文件。

现在 HelloWorld.html 文件应如下所示：

```
<html>
<head>
    <title>Hello World</title>
    <script type="text/javascript" src="AIRAliases.js"></script>
    <script type="text/javascript">
        function appLoad(){
            air.trace("Hello World");
        }
    </script>
</head>
<body onLoad="appLoad()">
    <h1>Hello World</h1>
</body>
</html>
```

## 测试应用程序

若要从命令行运行和测试应用程序，请使用 AIR Debug Launcher (ADL) 实用程序。 ADL 可执行程序可以在 AIR SDK 的 bin 目录中找到。如果您仍未安装 AIR SDK，请参阅第 16 页的 " 安装 AIR SDK"。

1 打开命令控制台或解释程序。转至为此项目创建的目录中。

2 运行以下命令：

```
adl HelloWorld-app.xml
```

AIR 窗口将会打开，以显示应用程序。同时，控制台窗口会显示由于 air.trace() 调用而产生的消息。

有关更多信息，请参阅第 154 页的 "AIR 应用程序描述符文件 "。

## 创建 AIR 安装文件

在成功运行应用程序后，可以使用 ADT 实用程序将应用程序打包到一个 AIR 安装文件中。AIR 安装文件是包含所有应用程序文件的存档文件，您可以将其分发给用户。必须先安装 Adobe AIR，然后才能安装打包的 AIR 文件。

为了确保应用程序安全，所有 AIR 安装文件必须经过数字签名。为便于开发，您可以使用 ADT 或其他证书生成工具生成一个基本的自签名证书。还可以从 VeriSign 或 Thawte 等商用证书颁发机构购买商用代码签名证书。用户安装自签名 AIR 文件时，发行商在安装过程中会显示为 " 未知 "。这是因为自签名证书仅确保 AIR 文件自创建后没有被更改。而无法阻止某人自签名一个伪装的 AIR 文件并将其显示为您的应用程序。对于公开发布的 AIR 文件，强烈建议使用可验证的商用证书。有关 AIR 安全问题的概述，请参阅 AIR 安全性 （针对 ActionScript 开发人员）或 AIR 安全性 （针对 HTML 开发人员）。

生成自签名证书和密钥对

❖ 从命令提示符处，输入以下命令（ADT 可执行文件位于 AIR SDK 的 bin 目录中）：

```
adt -certificate -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

ADT 会生成一个包含证书和相关私钥的名为 sampleCert.pfx 的 keystore 文件。

此示例使用了证书允许设置的最少数量的属性。密钥类型必须为 1024-RSA 或 2048-RSA（请参阅第 143 页的 " 对 AIR 应用程序进行签名 "）。

创建 **AIR** 安装文件

❖ 在命令提示符下，输入以下命令（在一行中）：

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air
HelloWorld-app.xml HelloWorld.html AIRAliases.js
```

系统将提示您输入 keystore 文件密码。

HelloWorld.air 参数表示 ADT 生成的 AIR 文件。 HelloWorld-app.xml 表示应用程序描述符文件。后面的参数表示应用程序所使用的文件。此示例仅使用了两个文件，但可以包含任意数量的文件和目录。 ADT 会验证主内容文件 HelloWorld.html 是否包含在包中，但是如果您忘记包含 AIRAliases.js，则应用程序根本无法工作。

在创建 AIR 包后，可以通过双击该包文件来安装和运行应用程序。也可在解释程序或命令窗口中键入 AIR 文件名作为命令。

## 后续步骤

在 AIR 中，通常 HTML 和 JavaScript 代码的行为与其在典型的 Web 浏览器中的行为相同。(事实上，AIR 和 Safari Web 浏览器都使用相同的 WebKit 呈现引擎。) 但是，如果在 AIR 中开发 HTML 应用程序，则必须了解一些重要差异。有关这些差异和其他重要主题的详细信息，请参阅 HTML 和 JavaScript 编程。

# 使用 Flex SDK 创建第一个桌面 AIR 应用程序

为了通过实际操作较快地说明 Adobe® AIR® 的工作原理，请按照以下说明使用 Flex SDK 创建一个简单的基于 SWF 的 AIR"Hello World" 应用程序。本教程说明如何使用随 Flex SDK 提供的命令行工具编译、测试和打包 AIR 应用程序（Flex SDK 包括 AIR SDK）。

首先必须安装运行时和 Adobe® Flex™。此教程使用 AMXMLC 编译器、 AIR 调试启动器 (ADL) 和 AIR 开发人员工具 (ADT)。可以在 Flex SDK 的 bin 目录中找到这些程序（请参阅第 17 页的 " 安装 Flex SDK"）。

## 创建 AIR 应用程序描述符文件

本节介绍如何创建应用程序描述符，它是具有以下结构的 XML 文件：

```
<application xmlns="...">
    <id>...</id>
    <versionNumber>...</versionNumber>
    <filename>…</filename>
    <initialWindow>
        <content>…</content>
        <visible>…</visible>
        <width>…</width>
        <height>…</height>
    </initialWindow>
</application>
```

**1** 创建名为 HelloWorld-app.xml 的 XML 文件，并将其保存到项目目录中。

2 添加 <application> 元素，并在其中包含 AIR 命名空间属性：

**<application xmlns="http://ns.adobe.com/air/application/2.7">** 命名空间的最后一段 “2.7” 指定应用程序所需的运行时版本。

3 添加 <id> 元素：

**<id>samples.flex.HelloWorld</id>** 应用程序 ID 用于唯一地标识应用程序及发行商 ID （通过 AIR 派生自用于对应用程序包进行签名的证书）。建议采用的形式为以点分隔的反向 DNS 样式的字符串，如 "com.company.AppName"。应用程序 ID 用于安装、访问专用应用程序文件系统存储目录、访问专用加密存储以及应用程序间的通信。

4 添加 <versionNumber> 元素：

**<versionNumber>1.0</versionNumber>** 可帮助用户确定安装哪个版本的应用程序。

注：如果您使用的是 AIR 2 或更早版本，则必须使用 <version> 元素，而非 <versionNumber> 元素。

5 添加 <filename> 元素：

**<filename>HelloWorld</filename>** 用作操作系统中应用程序可执行文件、安装目录和类似引用的名称。

6 添加包含下列子元素的 <initialWindow> 元素，从而为初始应用程序窗口指定属性：

**<content>HelloWorld.swf</content>** 标识 AIR 要加载的 SWF 根文件。

**<visible>true</visible>** 使窗口立即可见。

**<width>400</width>** 设置窗口宽度 （以像素为单位）。

**<height>200</height>** 设置窗口高度。

7 保存该文件。完整的应用程序描述符文件应如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
    <id>samples.flex.HelloWorld</id>
    <versionNumber>0.1</versionNumber>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>HelloWorld.swf</content>
        <visible>true</visible>
        <width>400</width>
        <height>200</height>
    </initialWindow>
</application>
```

此示例仅设置了几个可能的应用程序属性。有关应用程序属性的完整设置 （利用这些设置可以指定窗口镶边、窗口大小、透明度、默认安装目录、关联文件类型以及应用程序图标等），请参阅 第 154 页的 “AIR 应用程序描述符文件 ”

## 编写应用程序代码

注：基于 SWF 的 AIR 应用程序可以使用通过 MXML 或 Adobe® ActionScript® 3.0 定义的主类。此示例使用 MXML 文件定义其主类。使用 ActionScript 主类创建 AIR 应用程序的过程基本相同。不过不是将 MXML 文件编译成 SWF 文件，而是编译 ActionScript 类文件。使用 ActionScript 时，主类必须扩展 flash.display.Sprite。

与所有基于 Flex 应用程序类似，使用 Flex 框架构建的 AIR 应用程序包含一个主 MXML 文件。桌面 AIR 应用程序将 WindowedApplication 组件 （而不是 Application 组件）用作根元素。 WindowedApplication 组件提供用于控制应用程序及其初始窗口的属性、方法和事件。对于 AIR 不支持多窗口的平台和配置文件，请继续使用 Application 组件。在移动 Flex 应用程序中，还可以使用 View 或 TabbedViewNavigatorApplication 组件。

以下过程将创建 “Hello World” 应用程序：

1 使用文本编辑器创建名为 HelloWorld.mxml 的文件并添加以下 MXML 代码：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                       title="Hello World">
</s:WindowedApplication>
```

2 接下来，向应用程序添加 Label 组件（将其放置在 WindowedApplication 标记的内部）。

3 将 Label 组件的 text 属性设置为 "Hello AIR"。

4 将布局限制设置为始终居中。

以下示例显示到目前为止的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx"
                       title="Hello World">

    <s:Label text="Hello AIR" horizontalCenter="0" verticalCenter="0"/>
</s:WindowedApplication>
```

## 编译应用程序

在运行和调试应用程序之前，需要先使用 amxmlc 编译器将 MXML 代码编译成 SWF 文件。可以在 Flex SDK 的 bin 目录中找到 amxmlc 编译器。如果需要，可以将计算机的路径环境设置为包含 Flex SDK bin 目录。设置路径后，可以更方便地在命令行下运行实用程序。

1 打开命令解释程序或终端并导航到 AIR 应用程序的项目文件夹。

2 输入以下命令：

```
amxmlc HelloWorld.mxml
```

运行 amxmlc 会生成 HelloWorld.swf，它包含应用程序的编译代码。

注：如果应用程序无法编译，请修正语法或拼写错误。错误和警告显示在用于运行 amxmlc 编译器的控制台窗口中。

有关详细信息，请参阅第 117 页的 “为 AIR 编译 MXML 和 ActionScript 源文件 ”。

## 测试应用程序

若要从命令行运行和测试应用程序，请使用 AIR Debug Launcher (ADL) 启动使用其应用程序描述符文件的应用程序。（可以在 Flex SDK 的 bin 目录中找到 ADL。）

❖ 在命令提示符下，输入以下命令：

```
adl HelloWorld-app.xml
```

生成的 AIR 应用程序如下图所示：




通过使用 Label 控件的 horizontalCenter 和 verticalCenter 属性，文本位于窗口的中心位置。移动窗口或调整窗口大小，像对任何其他桌面应用程序所做的那样。

有关详细信息，请参阅 第 121 页的 "AIR Debug Launcher (ADL)"。

## 创建 AIR 安装文件

在成功运行应用程序后，可以使用 ADT 实用程序将应用程序打包到一个 AIR 安装文件中。AIR 安装文件是包含所有应用程序文件的存档文件，您可以将其分发给用户。必须先安装 Adobe AIR，然后才能安装打包的 AIR 文件。

为了确保应用程序安全，所有 AIR 安装文件必须经过数字签名。为便于开发，您可以使用 ADT 或其他证书生成工具生成一个基本的自签名证书。还可以从商用认证机构购买商用代码签名证书。用户安装自签名 AIR 文件时，发行商在安装过程中会显示为 " 未知 "。这是因为自签名证书仅确保 AIR 文件自创建后没有被更改。而无法阻止某人自签名一个伪装的 AIR 文件并将其显示为您的应用程序。对于公开发布的 AIR 文件，强烈建议使用可验证的商用证书。有关 AIR 安全问题的概述，请参阅 AIR 安全性 （针对 ActionScript 开发人员）或 AIR 安全性 （针对 HTML 开发人员）。

生成自签名证书和密钥对

❖ 在命令提示符下，输入以下命令 （可以在 Flex SDK 的 bin 目录中找到 ADT 可执行文件）：

```
adt -certificate -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

此示例使用了证书允许设置的最少数量的属性。密钥类型必须为 1024-RSA 或 2048-RSA （请参阅第 143 页的 " 对 AIR 应用程序进行签名 "）。

创建 **AIR** 包

❖ 在命令提示符下，输入以下命令 （在一行中）：

```
adt -package -storetype pkcs12 -keystore sampleCert.pfx HelloWorld.air
HelloWorld-app.xml HelloWorld.swf
```

系统将提示您输入 keystore 文件密码。键入密码并按 Enter。出于安全方面的考虑，不显示密码字符。

HelloWorld.air 参数表示 ADT 生成的 AIR 文件。 HelloWorld-app.xml 表示应用程序描述符文件。后面的参数表示应用程序所使用的文件。此示例只使用三个文件，但您可以包括任意数量的文件和目录。

在创建 AIR 包后，可以通过双击该包文件来安装和运行应用程序。也可在解释程序或命令窗口中键入 AIR 文件名作为命令。

有关详细信息，请参阅第 46 页的 " 对桌面 AIR 安装文件进行打包 "。

# 使用 Flex SDK 创建您的第一个 AIR for Android 应用程序

若要开始，必须已安装并设置 AIR 和 Flex SDK。此教程使用 Flex SDK 中的 AMXMLC 编译器和 AIR Debug Launcher (ADL)，以及 AIR SDK 中的 AIR Developer Tool (ADT)。请参阅第 17 页的 “ 安装 Flex SDK”。

同时还必须从 Android 网站下载并安装 Android SDK，如 Android 开发人员：安装 SDK 中所述。

注：有关 iPhone 开发的信息，请参阅使用 Flash Professional CS5 创建 Hello World iPhone 应用程序。

## 创建 AIR 应用程序描述符文件

本节介绍如何创建应用程序描述符，它是具有以下结构的 XML 文件：

```
<application xmlns="...">
    <id>...</id>
    <versionNumber>...</versionNumber>
    <filename>…</filename>
    <initialWindow>
        <content>…</content>
    </initialWindow>
    <supportedProfiles>...</supportedProfiles>
</application>
```

1 创建名为 HelloWorld-app.xml 的 XML 文件，并将其保存到项目目录中。

2 添加 <application> 元素，并在其中包含 AIR 命名空间属性：

**<application xmlns="http://ns.adobe.com/air/application/2.7">** 命名空间的最后一段 “2.7” 指定应用程序所需的运行时版本。

3 添加 <id> 元素：

**<id>samples.android.HelloWorld</id>** 应用程序 ID 与发行商 ID （由 AIR 派生自用于对应用程序包进行签名的证书）一起用于唯一地标识应用程序。建议采用的形式为以点分隔的反向 DNS 样式的字符串，如 "com.company.AppName"。

4 添加 <versionNumber> 元素：

**<versionNumber>0.0.1</versionNumber>** 可帮助用户确定安装哪个版本的应用程序。

5 添加 <filename> 元素：

**<filename>HelloWorld</filename>** 用作操作系统中应用程序可执行文件、安装目录和类似引用的名称。

6 添加包含下列子元素的 <initialWindow> 元素，从而为初始应用程序窗口指定属性：

**<content>HelloWorld.swf</content>** 标识 AIR 要加载的 HTML 根文件。

7 添加 <supportedProfiles> 元素。

**<supportedProfiles>mobileDevice</supportedProfiles>** 指定应用程序仅在移动配置文件中运行。

8 保存该文件。完整的应用程序描述符文件应如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
    <id>samples.android.HelloWorld</id>
    <versionNumber>0.0.1</versionNumber>
    <filename>HelloWorld</filename>
    <initialWindow>
        <content>HelloWorld.swf</content>
    </initialWindow>
    <supportedProfiles>mobileDevice</supportedProfiles>
</application>
```

此示例仅设置了几个可能的应用程序属性。应用程序描述符文件中还有其他可以使用的设置。例如，您可以将 <fullScreen>true</fullScreen> 添加到 initialWindow 元素以构建全屏应用程序。若要在 Android 上启用远程调试和访问控制功能，还必须将 Android 权限添加到应用程序描述符。此简单应用程序不需要权限，因此您现在不需要添加这些权限。

有关详细信息，请参阅第 60 页的 " 设置移动应用程序属性 "。

## 编写应用程序代码

创建名为 HelloWorld.as 的文件并使用文本编辑器添加以下代码：

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class HelloWorld extends Sprite
    {
        public function HelloWorld()
        {
            var textField:TextField = new TextField();
            textField.text = "Hello, World!";
            stage.addChild( textField );
        }
    }
}
```

## 编译应用程序

在运行和调试应用程序之前，需要先使用 amxmlc 编译器将 MXML 代码编译成 SWF 文件。可以在 Flex SDK 的 bin 目录中找到 amxmlc 编译器。如果需要，可以将计算机的路径环境设置为包含 Flex SDK bin 目录。设置路径后，可以更方便地在命令行下运行实用程序。

1 打开命令解释程序或终端并导航到 AIR 应用程序的项目文件夹。

2 输入以下命令：

```
amxmlc HelloWorld.as
```

运行 amxmlc 会生成 HelloWorld.swf，它包含应用程序的编译代码。

注：如果应用程序无法编译，请修正语法或拼写错误。错误和警告显示在用于运行 amxmlc 编译器的控制台窗口中。

有关详细信息，请参阅第 117 页的 " 为 AIR 编译 MXML 和 ActionScript 源文件 "。

## 测试应用程序

若要从命令行运行和测试应用程序，请使用 AIR Debug Launcher (ADL) 启动使用其应用程序描述符文件的应用程序。（可以在 AIR 和 Flex SDK 的 bin 目录中找到 ADL。）

❖ 在命令提示符下，输入以下命令：

```
adl HelloWorld-app.xml
```

有关详细信息，请参阅第 80 页的 " 使用 ADL 的设备模拟 "。

## 创建 APK 包文件

在成功运行应用程序后，可以使用 ADT 实用程序将应用程序打包到 AIR 包文件中。 APK 包文件是可以分发给用户的本机 Android 应用程序文件格式。

所有 Android 应用程序都必须进行签名。与 AIR 文件不同，它通常使用自签名的证书对 Android 应用程序进行签名。Android 操作系统不会尝试建立应用程序开发人员的身份。您可以使用由 ADT 生成的证书对 Android 包进行签名。用于提交到 Android Market 的应用程序的证书必须至少具有 25 年的有效期。

生成自签名证书和密钥对

❖ 在命令提示符下，输入以下命令 （可以在 Flex SDK 的 bin 目录中找到 ADT 可执行文件）：

```
adt -certificate -validityPeriod 25 -cn SelfSigned 1024-RSA sampleCert.pfx samplePassword
```

此示例使用了证书允许设置的最少数量的属性。密钥类型必须是 1024-RSA 或 2048-RSA （请参阅第 131 页的 “ADT certificate 命令 ”）。

创建 **AIR** 包

❖ 在命令提示符下，输入以下命令 （在一行中）：

```
adt -package -target apk -storetype pkcs12 -keystore sampleCert.p12 HelloWorld.apk HelloWorld-app.xml
HelloWorld.swf
```

系统将提示您输入 keystore 文件密码。键入密码并按 Enter。

有关详细信息，请参阅第 74 页的 “ 打包移动 AIR 应用程序 ”。

安装 **AIR** 运行时

您可以通过 Android Market 在您的设备上安装 AIR 运行时的最新版本。也可以在设备或 Android 仿真器上安装您的 SDK 中包括的运行时。

❖ 在命令提示符下，输入以下命令 （在一行中）：

```
adt -installRuntime -platform android -platformsdk
```

将 -platformsdk 标志设置为您的 Android SDK 目录 （指定工具文件夹的父级）。

ADT 会安装 SDK 中包括的 Runtime.apk。

有关详细信息，请参阅第 84 页的 “ 安装用于开发的 AIR 运行时和应用程序 ”。

安装 **AIR** 应用程序

❖ 在命令提示符下，输入以下命令 （在一行中）：

```
adt -installApp -platform android -platformsdk path-to-android-sdk -package path-to-app
```

将 -platformsdk 标志设置为您的 Android SDK 目录 （指定工具文件夹的父级）。

有关详细信息，请参阅第 84 页的 “ 安装用于开发的 AIR 运行时和应用程序 ”。

您可以通过点击设备或仿真器屏幕上的应用程序图标来启动应用程序。

# 第 6 章：开发针对桌面的 AIR 应用程序

## 开发桌面 AIR 应用程序的工作流程

开发 AIR 应用程序的基本工作流程和大多数传统开发模式是一样的：编码、编译、测试，并在开发周期即将结束时打包到安装程序文件中。

可以使用 Flash、Flex 和 ActionScript 来编写应用程序代码，使用 Flash Professional、Flash Builder 或 mxmlc 和 compc 命令行编译器进行编译。也可以使用 HTML 和 JavaScript 来编写应用程序代码，并跳过编译步骤。

可以使用 ADL 工具来测试桌面 AIR 应用程序，不需要事先打包和安装就可以直接运行应用程序。Flash Professional、Flash Builder、Dreamweaver 和 Aptana IDE 都是与 Flash 调试器集成的。当从命令行使用 ADL 时，也可以手动启动调试器工具 FDB。 ADL 自身会显示错误和 trace 语句输出。

必须将所有 AIR 应用程序打包到安装文件中。建议使用跨平台 AIR 文件格式，以下情况除外：

- 需要访问依赖于平台的 API，如 NativeProcess 类。
- 应用程序使用本机扩展。

在这种情况下，可以将 AIR 应用程序打包为特定于平台的本机安装程序文件。

### 基于 SWF 的应用程序

1. 编写 MXML 或 ActionScript 代码。
2. 创建需要的资源，例如图标位图文件。
3. 创建应用程序描述符。
4. 编译 ActionScript 代码。
5. 测试应用程序。
6. 使用 air 目标打包为 AIR 文件并进行签名。

### 基于 HTML 的应用程序

1. 编写 HTML 和 JavaScript 代码。
2. 创建需要的资源，例如图标位图文件。
3. 创建应用程序描述符。
4. 测试应用程序。
5. 使用 air 目标打包为 AIR 文件并进行签名。

### 为 AIR 应用程序创建本机安装程序

1. 编写代码 （ActionScript 或 HTML 和 JavaScript）。
2. 创建需要的资源，例如图标位图文件。
3. 创建应用程序描述符，从而指定 extendedDesktop 配置文件。
4. 编译任何 ActionScript 代码。
5. 测试应用程序。

6 使用 native 目标在每个目标平台上打包应用程序。

注：目标平台的本机安装程序必须在该平台上创建。例如，不能在 Mac 上创建 Windows 安装程序。可使用虚拟机 （如 VMWare）在同一计算机硬件上运行多个平台。

### 使用捕获的运行时捆绑创建 AIR 应用程序

1 编写代码 （ActionScript 或 HTML 和 JavaScript）。
2 创建需要的资源，例如图标位图文件。
3 创建应用程序描述符，从而指定 extendedDesktop 配置文件。
4 编译任何 ActionScript 代码。
5 测试应用程序。
6 使用 bundle 目标在每个目标平台上打包应用程序。
7 使用捆绑文件创建安装程序。（AIR SDK 不会提供用于创建此类安装程序的工具，但提供许多第三方工具包。）

注：目标平台的捆绑必须在该平台上创建。例如，不能在 Mac 上创建 Windows 捆绑。可使用虚拟机 （如 VMWare）在同一计算机硬件上运行多个平台。

## 设置桌面应用程序属性

设置应用程序描述符文件中的基本应用程序属性。本节涵盖了与桌面 AIR 应用程序相关的属性。第 154 页的 “AIR 应用程序描述符文件 ”全面描述了应用程序描述符文件的元素。

### 所需的 AIR 运行时版本

使用应用程序描述符文件的命名空间指定应用程序所需的 AIR 运行时版本。

在 application 元素中分配的命名空间，很大程度上决定了应用程序可以使用哪些功能。例如，如果应用程序使用 AIR 1.5 命名空间，而用户已经安装了 AIR 3.0，那么应用程序将参照 AIR 1.5 的行为 （即使该行为在 AIR 3.0 中发生了更改）。只有当您更改命名空间并发布更新时，应用程序才会访问新的行为和功能。安全性和 WebKit 更改是该策略的主要例外项。

使用根 application 元素的 xmlns 属性指定命名空间 :

```
<application xmlns="http://ns.adobe.com/air/application/3.0">
```

**更多帮助主题**

第 158 页的 “application”

### 应用程序标识

对于发布的每个应用程序，以下几个设置应该是唯一的。唯一的设置包括 ID、名称和文件名。

```
<id>com.example.MyApplication</id>
<name>My Application</name>
<filename>MyApplication</filename>
```

### 更多帮助主题

第 169 页的 “id”

第 165 页的 “filename”

第 176 页的 “name”

## 应用程序版本

在早于 AIR 2.5 的版本中，可以在 version 元素中指定应用程序。可以使用任何字符串。 AIR 运行时不会解释字符串； “2.0” 不是比 “1.0” 更高的版本。

```
<!-- AIR 2 or earlier -->
<version>1.23 Beta 7</version>
```

在 AIR 2.5 和更高版本中，可以在 versionNumber 元素中指定应用程序版本。不能再使用 version 元素。当为 versionNumber 指定值时，必须使用由点分隔的最多三个数字组成的序列，例如 :“0.1.2”。版本号的每段最多可以具有三个数字。 （即， “999.999.999” 是允许的最大版本号）。不必将所有三段都包含在号码中； “1” 和 “1.0” 都是合法的版本号。

也可以使用 versionLabel 元素来指定版本标签。添加版本标签时，在诸如应用程序安装程序对话框之类的位置中显示的是标签，而不是版本号。

```
<!-- AIR 2.5 and later -->
<versionNumber>1.23.7<versionNumber>
<versionLabel>1.23 Beta 7</versionLabel>
```

### 更多帮助主题

第 182 页的 “version”

第 183 页的 “versionLabel”

第 183 页的 “versionNumber”

## 主窗口属性

当 AIR 启动桌面上的应用程序时，会创建一个窗口，并在窗口中加载主 SWF 文件或 HTML 页。 AIR 使用 initialWindow 元素的子元素来控制该初始应用程序窗口的初始外观和行为。

- **content** — initalWindow 元素的 content 子级中的主应用程序 SWF 文件。当以桌面配置文件中的设备为目标时，可以使用 SWF 或 HTML 文件。

  ```
  <initialWindow>
      <content>MyApplication.swf</content>
  </initialWindow>
  ```

  必须包括 AIR 包中的文件 （使用 ADT 或 IDE）。如果只是引用应用程序描述符中的名称，不会自动将此文件包括进包中。
- **height** — 初始窗口的高度。
- **maximizable** — 是否显示最大化窗口的系统镶边。
- **maxSize** — 允许的最大尺寸。
- **minimizable** — 是否显示用于最小化窗口的系统镶边。
- **minSize** — 允许的最小尺寸。
- **renderMode** — 在 AIR 3 或更高版本中，对于桌面应用程序，渲染模式可设置为 auto、cpu、direct 或 gpu。在更早版本的 AIR 中，桌面平台上会忽略此设置。 renderMode 设置在运行时无法更改。
  - auto — 基本上与 cpu 模式相同。

- cpu — 呈现显示对象并将其复制到软件的显示内存中。仅在窗口处于全屏模式时，StageVideo 才可用。Stage3D 使用软件渲染器。
- direct — 运行时软件将呈现显示对象，但将呈现的帧复制到显示内存中 （块传输）采用硬件加速。 StageVideo 可用。Stage3D 使用硬件加速 （如果以其他方式可用）。如果窗口透明度设置为 true，则窗口 " 回退 " 到软件呈现和块传输。
- gpu — 保留供未来使用，不要在桌面应用程序中使用。

- **resizable** — 是否显示用于调整窗口大小的系统镶边。
- **systemChrome** — 是否使用标准操作系统窗口样式。窗口的 systemChrome 设置在运行时无法更改。
- **title** — 窗口的标题。
- **transparent** — 窗口是否是 alpha 混合的，以区别于背景。如果开启了透明度，窗口就不能使用系统镶边。窗口的透明设置在运行时无法更改。
- **visible** — 窗口是否在创建后即可见。默认情况下，窗口最初是不可见的，以便应用程序可以先绘制内容，然后再使其可见。
- **width** — 窗口的宽度。
- **x** — 窗口的水平位置。
- **y** — 窗口的垂直位置。

**更多帮助主题**

第 162 页的 "content"

第 168 页的 "height"

第 175 页的 "maximizable"

第 175 页的 "maxSize"

第 176 页的 "minimizable"

第 176 页的 "minimizable"

第 176 页的 "minSize"

第 178 页的 "renderMode"

第 179 页的 "resizable"

第 181 页的 "systemChrome"

第 182 页的 "title"

第 182 页的 "transparent"

第 183 页的 "visible"

第 184 页的 "width"

第 184 页的 "x"

第 185 页的 "y"

## 桌面功能

以下元素控制桌面安装和更新功能。

- customUpdateUI — 允许提供自己的对话框来更新应用程序。如果设置为 false （默认），则使用标准 AIR 对话框。

- fileTypes — 指定应用程序将注册为其默认打开程序的文件类型。如果其他应用程序已经是某种文件类型的默认打开程序，则 AIR 不会覆盖现有注册。但是，应用程序可以使用 NativeApplication 对象的 setAsDefaultApplication() 方法在运行时覆盖注册。在覆盖用户现有的文件类型关联前，最好征求用户同意。

  注：当将应用程序打包为捕获运行时捆绑（使用 -bundle 目标）时，会忽略文件类型注册。要注册指定文件类型，必须创建一个执行注册的安装程序。

- installFolder — 指定相对于标准应用程序安装文件夹（应用程序安装在其中）的路径。可以使用此设置来提供自定义文件夹名称和将多个应用程序组合在通用文件夹中。
- programMenuFolder — 为 Windows 的"所有程序"菜单指定菜单层次结构。可以使用该设置将多个应用程序组合在同一菜单中。如果未指定菜单文件夹，会直接将应用程序快捷方式添加到主菜单中。

**更多帮助主题**

第 163 页的 "customUpdateUI"

第 167 页的 "fileTypes"

第 172 页的 "installFolder"

第 177 页的 "programMenuFolder"

## 支持的配置文件

如果应用程序只有在桌面上才有意义，则可以防止将其安装在其他配置文件中的设备上，方法是通过将该配置文件从支持的配置文件列表中排除。如果应用程序使用 NativeProcess 类或本机扩展，则系统必须支持 extendedDesktop 配置文件。

如果应用程序描述符不包括 supportedProfile 元素，则会假定应用程序支持所有定义的配置文件。若要将应用程序限制在特定的配置文件列表中，请列出这些配置文件，用空格分隔：

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

有关 desktop 和 extendedDesktop 配置文件中支持的 ActionScript 类列表，请参阅第 187 页的 " 不同配置文件的功能 "。

**更多帮助主题**

第 180 页的 "supportedProfiles"

## 必需的本机扩展

支持 extendedDesktop 配置文件的应用程序可以使用本机扩展。

在应用程序描述符中声明 AIR 应用程序使用的所有本机扩展。下面的例子说明了用于指定两个所需本机扩展的语法：

```
<extensions>
    <extensionID>com.example.extendedFeature</extensionID>
    <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

extensionID 元素的值与扩展描述符文件中的 id 元素的值相同。扩展描述符文件是一个名为 extension.xml 的 XML 文件。已打包在从本机扩展开发人员处接收到的 ANE 文件中。

## 应用程序图标

在桌面上，会将应用程序描述符中指定的图标用作应用程序文件、快捷方式和程序菜单的图标。应用程序图标应作为 16x16、32x32、 48x48 和 128x128 像素的 PNG 图像集来提供。指定应用程序描述符文件的图标元素中图标文件的路径：

```
<icon>
    <image16x16>assets/icon16.png</image16x16>
    <image32x32>assets/icon32.png</image32x32>
    <image48x48>assets/icon48.png</image48x48>
    <image128x128>assets/icon128.png</image128x128>
</icon>
```

如果未提供指定尺寸的图标，则使用第二大尺寸并缩放至适合的大小。如果未提供任何图标，系统会使用默认系统图标。

**更多帮助主题**

第 168 页的 “icon”

第 169 页的 “imageNxN”

## 忽略的设置

桌面应用程序会忽略应用到移动配置文件功能的应用程序设置。忽略的设置包括：

- android
- aspectRatio
- autoOrients
- fullScreen
- iPhone
- renderMode （AIR 3 之前）
- requestedDisplayResolution
- softKeyboardBehavior

# 调试桌面 AIR 应用程序

如果使用 IDE （例如 Flash Builder、 Flash Professional 或 Dreamweaver）开发应用程序，调试工具通常都是内置的。以调试模式启动就可调试应用程序。如果不是使用支持直接调试的 IDE，可以使用 AIR Debug Launcher (ADL) 和 Flash Debugger (FDB) 来协助调试应用程序。

**更多帮助主题**

De Monster：Monster Debugger

第 218 页的 “ 使用 AIR HTML 内部检查器进行调试 ”

## 使用 ADL 运行应用程序

使用 ADL 可以无需打包和安装，直接运行 AIR 应用程序。如以下示例所示，将应用程序描述符文件作为参数传送给 ADL （必须首先编译应用程序中的 ActionScript 代码）：

```
adl myApplication-app.xml
```

ADL 会将 trace 语句、运行时异常和 HTML 分析错误输出到终端窗口。如果 FDB 进程正在等待传入连接，ADL 会连接到调试器。

也可使用 ADL 调试使用本机扩展的 AIR 应用程序。例如：

```
adl -extdir extensionDirs myApplication-app.xml
```

**更多帮助主题**

第 121 页的 “AIR Debug Launcher (ADL)”

## 输出 trace 语句

若要将 trace 语句输出到用于运行 ADL 的控制台，请使用 trace() 函数将 trace 语句添加到代码中。

ActionScript 示例：

```
//ActionScript
trace("debug message");
```

JavaScript 示例：

```
//JavaScript
air.trace("debug message");
```

在 JavaScript 代码中，可使用 alert() 和 confirm() 函数来显示应用程序中的调试消息。此外，语法错误的行号以及任何未捕获的 JavaScript 异常均输出到该控制台。

注：若要使用 JavaScript 示例中显示的 air 前缀，则必须将 AIRAliases.js 文件导入到此页。此文件位于 AIR SDK 的 frameworks 目录中。

## 连接 Flash Debugger (FDB)

若要使用 Flash Debugger 调试 AIR 应用程序，请启动 FDB 会话然后使用 ADL 启动您的应用程序。

注：在基于 SWF 的 AIR 应用程序中，必须使用 -debug 标志编译 ActionScript 源文件。（在 Flash Professional 中，选中 “ 发布设置 ” 对话框中的 “ 允许调试 ” 选项。）

1 启动 FDB。可以在 Flex SDK 的 bin 目录中找到 FDB 程序。

   控制台显示 FDB 提示符：<fdb>

2 执行 run 命令：<fdb>run [Enter]

3 在不同命令或解释程序控制台中，启动应用程序的调试版本：

   ```
   adl myApp.xml
   ```

4 使用 FDB 命令根据需要设置断点。

5 键入：continue [Enter]

如果 AIR 应用程序基于 SWF，则调试器只控制 ActionScript 代码的执行。如果 AIR 应用程序基于 HTML，则调试器只控制 JavaScript 代码的执行。

若要在不连接到调试器的情况下运行 ADL，请加入 -nodebug 选项：

```
adl myApp.xml -nodebug
```

有关 FDB 命令的基本信息，请执行 help 命令：

```
<fdb>help [Enter]
```

有关 FDB 命令的详细信息，请参阅 Flex 文档中的使用命令行调试器命令。

# 对桌面 AIR 安装文件进行打包

每个 AIR 应用程序必须至少包含一个应用程序描述符文件和主 SWF 或 HTML 文件。任何随应用程序安装的其他资源也必须打包在 AIR 文件中。

本文讨论了使用 SDK 随附的命令行工具对 AIR 应用程序进行打包。有关使用 Adobe 创作工具对应用程序进行打包的信息，请参阅以下内容：

- Adobe® Flex® Builder™，请参阅使用 Flex Builder 打包 AIR 应用程序。
- Adobe® Flash® Builder™，请参阅使用 Flash Builder 打包 AIR 应用程序。
- Adobe® Flash® Professional，请参阅为 Adobe AIR 发布。
- Adobe® Dreamweaver®，请参阅在 Dreamweaver 中创建 AIR 应用程序。

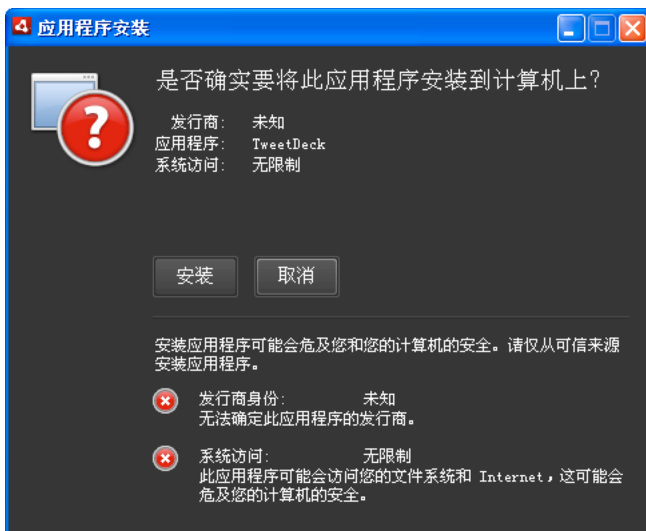
必须使用数字证书对所有 AIR 安装程序文件签名。 AIR 安装程序使用该签名验证应用程序文件自签名之后是否未发生任何更改。可以使用证书颁发机构颁发的代码签名证书，也可以使用自签名证书。

当使用由受信任证书颁发机构颁发的证书时，因为您的身份是发布者，这就为应用程序的用户提供了一些保证。安装对话框反映了您的身份已经过证书颁发机构验证这一事实：




由受信任证书签名的应用程序的安装确认对话框

当您使用自签名证书时，用户无法验证您的身份是否为签名者。自签名证书不能完全保证包未经过修改。（这是因为合法的安装文件在到达用户之前可能被假冒文件替代。）安装对话框反映了发布者的身份无法验证这一事实。用户在安装您的应用程序时将承担更大的安全风险：




由自签名证书签名的应用程序的安装确认对话框

使用 ADT -package 命令可以在一个步骤中对 AIR 文件进行打包和签名。您还可以使用 -prepare 命令创建一个未签名的中间包，然后在单独的步骤中使用 -sign 命令对中间包签名。

注: Java 1.5 版及更高版本不允许在用于保护 PKCS12 证书文件的密码中使用高位 ASCII 字符。当您创建或导出代码签名的证书文件时，仅在密码中使用常规 ASCII 字符。

当对安装包签名时， ADT 将自动连接时间戳签发机构服务器以验证时间。时间戳信息包含在 AIR 文件中。您可以在将来任一时间安装包含已经过验证的时间戳的 AIR 文件。如果 ADT 无法连接到时间戳服务器，则取消打包。您可以覆盖时间戳设置选项，但是如果没有时间戳，在对安装文件签名所使用的证书过期后， AIR 应用程序将停止安装。

如果您正在创建包以更新现有的 AIR 应用程序，必须使用与原始应用程序相同的证书对包进行签名。如果原始证书已续签或已在过去 180 天内过期，或者您要更改为新的证书，则可以申请迁移签名。迁移签名涉及到使用新证书和旧证书对应用程序 AIR 文件进行签名。使用 -migrate 命令可以应用迁移签名 （如 第 130 页的 “ADT migrate 命令 ” 中所述） 。

重要说明：在原始证书过期后，在严格的 180 天宽限期内可以申请迁移签名。如果没有迁移签名，现有用户必须卸载其现有应用程序才能安装新版本。宽限期仅适用于在应用程序描述符命名空间中指定 AIR 1.5.3 版或更高版本的应用程序。如果目标是 AIR 运行时的早期版本，则没有宽限期。

AIR 1.1 之前的版本不支持迁移签名。您必须使用 SDK 1.1 或更高版本打包应用程序才能申请迁移签名。

使用 AIR 文件部署的应用程序称为桌面配置文件应用程序。如果应用程序描述符文件不支持桌面配置文件，则不能使用 ADT 为 AIR 应用程序打包本机安装程序。可以使用应用程序描述符文件中的 supportedProfiles 元素限制此配置文件。请参阅第 186 页的 “ 设备配置文件 ” 和 第 180 页的 “supportedProfiles”。

注：应用程序描述符文件中的设置确定 AIR 应用程序的标识及其默认安装路径。请参阅 第 154 页的 “AIR 应用程序描述符文件 ”。

### 发行商 ID

从 AIR 1.5.3 开始，将弃用发行商 ID。新应用程序 （最初使用 AIR 1.5.3 或更高版本发布）不需要也不应指定发行商 ID。

当更新使用 AIR 的早期版本发布的应用程序时，您必须在应用程序描述符文件中指定原始发行商 ID。否则，会将应用程序的安装版本和更新版本视为不同的应用程序。如果您使用其他 ID 或省略 publisherID 标签，用户必须在安装新版本之前卸载早期版本。

要确定原始发行商 ID，请在安装原始应用程序的 META-INF/AIR 子目录中查找 publisherid 文件。此文件中的字符串就是发行商 ID。要手动指定发行商 ID，应用程序描述符必须在应用程序描述符文件的命名空间声明中指定 AIR 1.5.3 运行时 （或更高版本） 。

对于在 AIR 1.5.3 之前发布的应用程序 （在应用程序描述符命名空间中指定 AIR 的早期版本时，则是使用 AIR 1.5.3 SDK 发布的应用程序） ，将根据签名证书计算发行商 ID。将此 ID 与应用程序 ID 一起使用来确定应用程序的标识。发行商 ID （如果存在）用于以下用途：

- 验证 AIR 文件是一个更新而不是要安装的新应用程序
- 作为加密本地存储加密密钥的一部分
- 作为应用程序存储目录路径的一部分
- 作为本地连接的连接字符串的一部分
- 作为用于使用 AIR 浏览器内 API 来调用应用程序的标识字符串的一部分
- 作为 OSID （在创建自定义安装 / 卸载程序时使用）的一部分

在 AIR 1.5.3 之前，如果您使用新的或续签的证书对包含迁移签名的应用程序更新进行签名，则可以更改应用程序的发行商 ID。当发行商 ID 改变时，所有依赖该 ID 的 AIR 功能的行为也会改变。例如，将无法访问现有加密本地存储中的数据，所有创建到应用程序的本地连接的 Flash 或 AIR 实例必须使用连接字符串中的新 ID。

在 AIR 1.5.3 或更高版本中，发行商 ID 不是基于签名证书的，而且只能在应用程序描述符中包含 publisherID 标签的情况下指定。如果为更新 AIR 包指定的发行商 ID 与其当前的发行商 ID 不匹配，则无法更新应用程序。

## 使用 ADT 打包

可以使用 AIR ADT 命令行工具对 AIR 应用程序进行打包。打包前必须先编译所有的 ActionScript、 MXML 和任何扩展代码。还必须有代码签名证书。

有关 ADT 命令和选项的详细参考，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。

创建 **AIR** 包

若要创建 AIR 包，请使用 ADT package 命令，同时针对发行版将目标类型设置为 air。

```
adt -package -target air -storetype pkcs12 -keystore ../codesign.p12 myApp.air myApp-app.xml myApp.swf icons
```

此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 “ 路径环境变量 ”。）

必须从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、myApp.swf 和图标目录。

当运行如上所示的命令时，ADT 会提示输入 keystore 密码。（键入的密码字符不会总是显示；只需在键入结束后按 Enter。）

从 **AIRI** 文件创建 **AIR** 包

您可以对 AIRI 文件进行签名，以创建可安装的 AIR 包：

```
adt -sign -storetype pkcs12 -keystore ../codesign.p12 myApp.airi myApp.air
```

# 对桌面本机安装程序进行打包

从 AIR 2 开始，可以使用 ADT 创建本机应用程序安装程序，以用于分布 AIR 应用程序。例如，您可以在 Windows 中构建一个 EXE 安装程序文件，用来分发 AIR 应用程序。您可以在 Mac OS 中构建一个 DMG 安装程序文件，用来分发 AIR 应用程序。在 AIR 2.5 和 AIR 2.6 中，您可以在 Linux 中构建一个 DEB 或 RPM 安装程序文件，用来分发 AIR 应用程序。

随本机应用程序安装程序安装的应用程序称为扩展的桌面配置文件应用程序。如果应用程序描述符文件不支持桌面扩展配置文件，则不能使用 ADT 为 AIR 应用程序打包本机安装程序。可以使用应用程序描述符文件中的 supportedProfiles 元素限制此配置文件。请参阅第 186 页的 “ 设备配置文件 ” 和 第 180 页的 “supportedProfiles”。

您可以通过两种基本方式构建 AIR 应用程序的本机安装程序版本：

- 可以根据应用程序描述符文件及其他源文件构建本机安装程序。（其他源文件可能包括 SWF 文件、 HTML 文件及其他资源。）
- 还可以根据 AIR 文件或 AIRI 文件构建本机安装程序。

要在其中使用 ADT 的操作系统必须与要生成本机安装程序文件的操作系统相同。因此，要为 Windows 创建 EXE 文件，请在 Windows 中运行 ADT。要为 Mac OS 创建 DMG 文件，请在 Mac OS 中运行 ADT。要为 Linux 创建 DEB 或 RPG 文件，请在 Linux 中通过 AIR 2.6 SDK 运行 ADT。

在创建本机安装程序以分发 AIR 应用程序时，该应用程序将获得下列功能：

- 可以使用 NativeProcess 类启动并与本机进程交互。有关详细信息，请参阅下列内容之一：
  - 与 AIR 中的本机进程通信 （针对 ActionScript 开发人员）
  - 与 AIR 中的本机进程通信 （针对 HTML 开发人员）
- 可使用本机扩展。
- 它可以使用 File.openWithDefaultApplication() 方法，打开任何默认系统应用程序定义为将其打开的文件，而无论文件为哪种类型。（对没有随本机安装程序安装的应用程序具有限制。有关详细信息，请参阅语言参考中的 File.openWithDefaultApplication() 条目。）

但是，当将其打包成本机安装程序时，应用程序会失去 AIR 文件格式的某些优点。单个文件不能再分发到所有台式计算机。内置更新功能 （和 updater 框架）无效。

用户双击本机安装程序文件时，将安装 AIR 应用程序。如果计算机中尚未安装所需的 Adobe AIR 版本，安装程序将从网络下载该版本并首先安装。如果没有获取正确的 Adobe AIR 版本的网络连接 （如有必要），则安装将失败。此外，如果 Adobe AIR 2 中不支持该操作系统，则安装将失败。

注： 如果希望在已安装的应用程序中可以执行某个文件，请确保在打包应用程序时该文件在文件系统上可执行。（在 Mac 和 Linux 上，如果需要，可以使用 chmod 来设置可执行标记。）

### 从应用程序源文件创建一个本机安装程序

要在应用程序的源文件以外构建一个本机安装程序，请在一个命令行中使用带有以下语法的 -package 命令：

```
adt -package AIR_SIGNING_OPTIONS
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    app_xml
    [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

此语法与用于打包 AIR 文件 （不包括本机安装程序）的语法类似。但存在几点差异：

- 可以将 -target native 选项添加到命令中。（如果指定 -target air，则 ADT 生成 AIR 文件，而不是本机安装程序文件。）
- 可以将目标 DMG 或 EXE 文件指定为 installer_file。
- 或者，您可以在 Windows 上再添加一组签名选项，如语法清单中的 [WINDOWS_INSTALLER_SIGNING_OPTIONS] 所指示的那样。在 Windows 中，除了对 AIR 文件进行签名，还可以对 Windows 安装程序文件进行签名。使用与对 AIR 文件进行签名相同类型的证书和签名选项语法 （请参阅第 135 页的 “ADT 代码签名选项 ”）。可以使用相同的证书对 AIR 文件和安装程序文件进行签名，也可以指定不同的证书。当用户从 Web 下载已签名的 Windows 安装程序文件时，Windows 将根据证书标识文件源。

有关除 -target 选项之外的其他 ADT 选项的详细信息，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。

以下示例创建 DMG 文件 （Mac OS 的本机安装程序文件）：

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    index.html resources
```

以下示例创建 EXE 文件 （Windows 的本机安装程序文件）：

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.exe
    application.xml
    index.html resources
```

以下示例创建 EXE 文件并对其进行签名：

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    -storetype pkcs12
    -keystore myCert.pfx
    myApp.exe
    application.xml
    index.html resources
```

### 为使用本机扩展的应用程序创建本机安装程序

您可以在源文件以外为应用程序及应用程序所需的本机扩展包构建本机安装程序。在一个命令行中采用以下语法使用 -package 命令：

```
adt -package AIR_SIGNING_OPTIONS
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    app_xml
    -extdir extension-directory
    [file_or_dir | -C dir file_or_dir | -e file dir ...] ...
```

此语法与用于打包本机安装程序的语法相同，具有一个附加选项。使用 -extdir ***extension-directory*** 选项可指定包含应用程序使用的 ANE 文件 （本机扩展）的目录。

有关 ADT 选项的详细信息，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。

下面的示例为使用本机扩展的应用程序创建一个 DMG 文件 （用于 Mac OS 的本机安装程序文件）：

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    -extdir extensionsDir
    index.html resources
```

### 从 **AIR** 文件或 **AIRI** 文件创建本机安装程序

可使用 ADT 基于 AIR 文件或 AIRI 文件生成本机安装程序文件。要基于 AIR 文件构建本机安装程序，请在一个命令行中使用带有以下语法的 ADT -package 命令：

```
adt -package
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    air_file
```

此语法与用于根据 AIR 应用程序的源文件创建本机安装程序的语法类似。但存在几点差异：

- 为 AIR 应用程序指定一个 AIR 文件 （而不是应用程序描述符文件或其他源文件）作为源。
- 不要为 AIR 文件指定签名选项，因为已对该文件进行签名

要基于 AIRI 文件构建本机安装程序，请在一个命令行中使用带有以下语法的 ADT -package 命令：

```
adt AIR_SIGNING_OPTIONS
    -package
    -target native
    [WINDOWS_INSTALLER_SIGNING_OPTIONS]
    installer_file
    airi_file
```

此语法与用于根据 AIR 文件创建本机安装程序的语法类似。但存在几点差异：

- 指定一个 AIRI 文件作为源。
- 为目标 AIR 应用程序指定签名选项。

以下示例基于 AIR 文件创建 DMG 文件 （Mac OS 的本机安装程序文件）：

```
adt -package -target native myApp.dmg myApp.air
```

以下示例基于 AIR 文件创建 EXE 文件 （Windows 的本机安装程序文件）：

```
adt -package -target native myApp.exe myApp.air
```

以下示例基于 AIR 文件创建 EXE 文件并对其进行签名:

```
adt -package -target native -storetype pkcs12 -keystore myCert.pfx myApp.exe myApp.air
```

以下示例基于 AIRI 文件创建 DMG 文件 （Mac OS 的本机安装程序文件）：

```
adt -storetype pkcs12 -keystore myCert.pfx -package -target native myApp.dmg myApp.airi
```

以下示例基于 AIRI 文件创建 EXE 文件 （Windows 的本机安装程序文件）：

```
adt -storetype pkcs12 -keystore myCert.pfx -package -target native myApp.exe myApp.airi
```

以下示例创建一个 EXE 文件 （基于 AIRI 文件）并使用 AIR 和本机 Windows 签名对该文件进行签名:

```
adt -package -storetype pkcs12 -keystore myCert.pfx -target native -storetype pkcs12 -keystore myCert.pfx
myApp.exe myApp.airi
```

# 为桌面计算机打包捕获运行时捆绑

捕获运行时捆绑是一个包含应用程序代码以及运行时专用版本的包。以此方式打包的应用程序使用捆绑的运行时，而不是安装在用户计算机上其他位置的共享运行时。

生成的捆绑在 Windows 上是应用程序文件的自包含文件夹，在 Mac OS 上是一个 .app 捆绑。在目标操作系统下运行时，必须为该操作系统生成捆绑。（虚拟机，如 VMWare，可用于在一台计算机上运行多个操作系统。）

不必进行安装即可从该文件夹或捆绑运行应用程序。

优点

- 生成自包含应用程序
- 安装无需进行 Internet 访问
- 安装程序与运行时更新隔离
- 企业可以认证特定应用程序和运行时组合
- 支持传统软件部署模型
- 无需单独的运行时重新分发
- 可使用 NativeProcess API
- 可使用本机扩展
- 可使用 File.openWithDefaultApplication() 函数，无限制
- 可从 USB 或光盘运行，无需安装

缺点

- 当 Adobe 发布安全修补程序时，不会自动向用户提供关键安全修补程序
- 无法使用 .air 文件格式
- 必须创建自己的安装程序 （若需要）
- 不支持 AIR 更新和 API 框架
- 不支持用于从网页安装和启动 AIR 应用程序的 AIR 浏览器内 API
- 在 Windows 上，必须由安装程序处理文件注册
- 需要更大的应用程序磁盘空间

## 在 Windows 上创建捕获运行时捆绑

要为 Windows 创建捕获运行时捆绑，必须在 Windows 操作系统下运行时打包应用程序。使用 ADT bundle 目标打包应用程序：

```
adt -package
    -keystore ..\cert.p12 -storetype pkcs12
    -target bundle
    myApp
    myApp-app.xml
    myApp.swf icons resources
```

此命令在名为 “myApp” 的目录中创建捆绑。目录包含应用程序的文件以及运行时文件。可直接从文件夹运行程序。不过，要创建一个程序菜单项、注册文件类型或 URI 方案处理函数，您必须创建一个设置必需的注册表项的安装程序。 AIR SDK 不包含用于创建此类安装程序的工具，但提供几个第三方选项，包括商用和免费开放源安装程序工具包。

您可在 WIndows 上对本机可执行文件进行签名，方法是在命令行上的 -target bundle 条目后面另外指定一组签名选项。这些签名选项标识应用本机 Windows 签名时使用的私钥和关联证书。（通常可使用 AIR 代码签名证书。）仅签名主要可执行文件。此进程不会对随应用程序打包的任何其他可执行文件进行签名。

### 文件类型关联

要在 Windows 上将应用程序与公共或自定义文件类型相关联，安装程序必须设置相应的注册表项。应用程序描述符文件的 fileTypes 元素中也应列出文件类型。

有关 Windows 文件类型的更多信息，请参阅 MSDN Library: File Types and File Associations

### URI 处理函数注册

要使应用程序采用指定 URI 方案处理 URL 的启动，安装程序必须设置必需的注册表项。

有关注册应用程序以处理 URI 方案的更多信息，请参阅 MSDN Library: Registering an Application to a URL Protocol

## 在 Mac OS X 上创建捕获运行时捆绑

要为 Mac OS X 创建捕获运行时捆绑，必须在 Macintosh 操作系统下运行时打包应用程序。使用 ADT bundle 目标打包应用程序：

```
adt -package
    -keystore ../cert.p12 -storetype pkcs12
    -target bundle
    myApp.app
    myApp-app.xml
    myApp.swf icons resources
```

此命令创建名为 “myApp.app” 的应用程序捆绑。捆绑包含应用程序的文件以及运行时文件。双击 myApp.app 图标可运行应用程序，将其拖放到合适的位置 （如 Applications 文件夹）可进行安装。不过，要注册文件类型或 URI 方案处理函数，必须编辑应用程序包内的属性列表文件。

要进行分发，可创建一个磁盘映像文件 (.dmg)。 Adobe AIR SDK 不会提供用于为捕获运行时捆绑创建 dmg 文件的工具。

### 文件类型关联

要在 Mac OS X 上将应用程序与公共或自定义文件类型相关联，必须编辑捆绑中的 info.plist 文件以设置 CFBundleDocumentTypes 属性。请参阅 Mac OS X Developer Library: Information Property List Key Reference, CFBundleURLTypes。

### URI 处理函数注册

要使应用程序采用指定 URI 方案处理 URL 的启动，必须编辑捆绑中的 info.plist 文件以设置 CFBundleURLTypes 属性。请参阅 Mac OS X Developer Library: Information Property List Key Reference, CFBundleDocumentTypes。

# 针对桌面计算机分发 AIR 包

AIR 应用程序可以作为 AIR 包来分发，该包包含应用程序代码和所有资源。可以通过通常采用的任何方法来分发此包，例如，通过下载、通过电子邮件或通过物理介质 （如 CD-ROM）。用户可以通过双击 AIR 文件来安装此应用程序。可以使用 AIR 浏览器内 API（基于 Web 的 ActionScript 库）让用户安装 AIR 应用程序（如果需要，也可以安装 Adobe® AIR®），方法是单击网页上的某个链接。

也可以将 AIR 应用程序作为本机安装程序来进行打包和分发（即，可以作为 Windows 中的 EXE 文件、 Mac 中的 DMG 文件和 Linux 中的 DEB 或 RPM 文件）。可以根据相关的平台惯例，对本机安装包进行分发和安装。将应用程序作为本机包来分发时，会失去 AIR 文件格式的某些优点。即，不能再在大部分平台中使用单个安装文件，不能再使用 AIR 更新框架，也不能再使用浏览器内 API。

## 在桌面上安装并运行 AIR 应用程序

您可以直接将 AIR 文件发送给接收方。例如，您可以将 AIR 文件作为电子邮件附件或作为网页中的链接发送。

用户下载 AIR 应用程序后，就会按以下说明安装该应用程序：

**1** 双击 AIR 文件。

计算机上必须安装了 Adobe AIR。

**2** 在 " 安装 " 窗口中，保留默认设置处于选定状态不变，然后单击 " 继续 "。

在 Windows 中， AIR 会自动执行以下操作：

- 将此应用程序安装到 Program Files 目录
- 为此应用程序创建一个桌面快捷方式
- 创建 " 开始 " 菜单快捷方式
- 在 " 添加 / 删除程序 " 控制面板中添加一个应用程序条目

在 Mac OS 中，默认情况下，会将此应用程序添加到 Applications 目录中。

如果已安装此应用程序，安装程序将让用户选择是打开此应用程序的现有版本还是更新到所下载 AIR 文件中的版本。安装程序使用 AIR 文件中的应用程序 ID 和发行商 ID 来标识此应用程序。

**3** 安装完成后，单击 " 完成 "。

在 Mac OS 中，若要安装某一应用程序的更新版本，用户需要具有足够的系统权限才能将新版本安装到应用程序目录中。在 Windows 和 Linux 中，用户需要具有管理权限。

应用程序也可以通过 ActionScript 或 JavaScript 安装新版本。有关详细信息，请参阅第 197 页的 " 更新 AIR 应用程序 "。

安装 AIR 应用程序后，用户只需双击此应用程序的图标即可运行它，就像任何其他桌面应用程序一样。

- 在 Windows 中，请双击该应用程序的图标 （安装在桌面上或文件夹中），或者从 " 开始 " 菜单中选择该应用程序。
- 在 Linux 中，请双击该应用程序的图标 （安装在桌面上或文件夹中），或者从应用程序菜单中选择该应用程序。
- 在 Mac OS 中，请在该应用程序的安装文件夹中双击该应用程序。默认安装目录为 /Applications 目录。

注：只有为 AIR 2.6 或更早版本开发的 AIR 应用程序可安装在 Linux 上。

使用 AIR 无缝安装 功能，用户可通过单击网页中的链接来安装 AIR 应用程序。使用 AIR 浏览器调用 功能，用户可以通过单击网页中的链接来运行安装的 AIR 应用程序。下一节中介绍了这些功能。

## 从网页中安装和运行桌面 AIR 应用程序

AIR 浏览器内 API 使您可以从网页中安装和运行 AIR 应用程序。AIR 浏览器内 API 由 Adobe 承载的 SWF 库 air.swf 提供。AIR SDK 包括使用此库安装、更新或启动 AIR 应用程序（如果需要，还有运行时）的示例"标志"应用程序。可以修改提供的示例标志或创建自己的标志 Web 应用程序，该应用程序直接使用联机 air.swf 库。

可以通过网页标志安装任何 AIR 应用程序。但是，只有应用程序描述符文件中包括 <allowBrowserInvocation>true</allowBrowserInvocation> 元素的应用程序才可以通过网页标志来启动。

**更多帮助主题**

第 190 页的"AIR.SWF 浏览器内 API"

通过 Web 分发 AIR 应用程序

## 在桌面计算机上安装日志

IT 管理员可以使用标准的桌面部署工具以静默方式安装 Adobe AIR 运行时和 AIR 应用程序。 IT 管理员可以执行以下操作：

- 使用工具（例如， Microsoft SMS、 IBM Tivoli，或任何允许使用引导程序进行静默安装的部署工具）以静默方式安装 Adobe AIR 运行时
- 使用用于部署运行时的相同工具以静默方式安装 AIR 应用程序

有关详细信息，请参阅 Adobe AIR 管理员指南 (http://www.adobe.com/go/learn_air_admin_guide_cn)。

## 桌面计算机上的安装日志

安装 AIR 运行时本身或 AIR 应用程序时都会记录安装日志。您可以通过查看日志文件帮助确定所有安装或更新问题的发生原因。

日志文件在以下位置创建：

- Mac：标准系统日志 (/private/var/log/system.log)

  可以通过打开控制台应用程序（通常在实用程序文件夹中可以找到）来查看 Mac 系统日志。
- Windows XP：C:\Documents and Settings\<username>\Local Settings\Application Data\Adobe\AIR\logs\Install.log
- Windows Vista 和 Windows 7：C:\Users\<username>\AppData\Local\Adobe\AIR\logs\Install.log
- Linux：/home/<username>/.appdata/Adobe/AIR/Logs/Install.log

注：在 AIR 2 之前的 AIR 版本中不创建这些日志文件。

# 第 7 章：开发针对移动设备的 AIR 应用程序

移动设备上的 AIR 应用程序会作为本机应用程序进行部署。它们使用设备的应用程序格式，而不是 AIR 文件格式。当前 AIR 支持 Android APK 包和 iOS IPA 包。在创建应用程序包的发行版本之后，您可以通过标准平台机制来分发应用程序。对于 Android，这通常表示 Android Market；对于 iOS，则表示 Apple 应用程序库。

您可以使用 AIR SDK 和 Flash Professional、Flash Builder 或另外的 ActionScript 开发工具，为移动设备构建 AIR 应用程序。当前不支持基于 HTML 的移动 AIR 应用程序。

注：Research In Motion (RIM) BlackBerry Playbook 提供自己的用于 AIR 开发的 SDK。有关 Playbook 开发的信息，请参阅 RIM：BlackBerry Tablet OS 开发。

注：本文档介绍如何使用 AIR 2.6 SDK 或更高版本开发 iOS 应用程序。使用 AIR 2.6+ 创建的应用程序可在运行 iOS 4 或更高版本的 iPhone 3G、iPhone 4 和 iPad 设备上安装。若要为早期版本的 iOS 开发 AIR 应用程序，您必须按照构建 iPhone 应用程序中的说明使用 AIR 2 Packager for iPhone。

有关运行 AIR 应用程序的完整系统要求，请参阅 Adobe AIR 系统要求。

## 设置开发环境

除了一般的 AIR、Flex 和 Flash 开发环境设置之外，移动平台还有其他设置要求。（有关设置基本 AIR 开发环境的更多信息，请参阅第 16 页的 " 适用于 AIR 开发的 Adobe Flash Platform 工具 "。）

### Android 设置

AIR 2.6+ 中的 Android 通常不需要特殊设置。 Android ADB 工具包含在 AIR SDK 中 （在 lib/android/bin 文件夹中）。AIR SDK 使用 ADB 工具在设备上安装、卸载并运行应用程序包。您还可以使用 ADB 查看系统日志。若要创建并运行 Android 模拟器，您必须下载单独的 Android SDK。

如果您的应用程序向应用程序描述符中的 <manifestAdditions> 元素添加元素，而当前版本 AIR 认为这些元素无效，那么您必须安装更新版本的 Android SDK。将 AIR_ANDROID_SDK_HOME 环境变量或 -platformsdk 命令行参数设置为 SDK 的文件路径。 AIR 打包工具，即 ADT，使用此 SDK 验证 <manifestAdditions> 元素中的条目。

在 AIR 2.5 中，您必须从 Google 下载 Android SDK 的单独副本。您可以将 AIR_ANDROID_SDK_HOME 环境变量设置为引用 Android SDK 文件夹。如果没有设置此环境变量，则必须在 ADT 命令行的 -platformsdk 参数中指定指向 Android SDK 的路径。

#### 更多帮助主题

第 141 页的 "ADT 环境变量 "

第 237 页的 " 路径环境变量 "

### iOS 设置

但是，若要在设备上安装和测试 iOS 应用程序并分发该应用程序，您必须加入 Apple iOS 开发人员计划 （这是一项免费服务）。加入 iOS 开发人员计划之后，可以访问 iOS Provisioning Portal，通过该网站，可以从 Apple 获得在设备上安装应用程序所需的以下项目和文件，以供测试和后续分发之用。这些项目和文件包括：

- 开发和分发证书
- 应用程序 ID

- 开发和分发供给文件

# 移动应用程序设计注意事项

移动设备的操作上下文和物理特性要求仔细进行编码和设计。例如，必须要简化代码以实现尽可能最快的执行速度。当然，代码优化只能起到这么大的作用；在设备限制范围内发挥作用的智能设计还有助于防止可视演示使呈现系统负担过重。

代码

尽管使代码更快地运行始终是有用的，但是大多数移动设备的处理器速度较慢，所以就更值得花费一些时间编写精简代码。此外，移动设备几乎总是使用电池电能运行。如果用较少的工作量达到相同的效果，则会使用较少的电池电量。

设计

在设计应用程序的用户体验时，必须考虑小屏幕尺寸、触屏交互模式甚至不断变化的移动用户环境等因素。

代码和设计一起

如果您的应用程序使用动画，则呈现优化非常重要。但是，只有代码优化通常是不够的。在设计应用程序可视效果方面，必须让代码可以将它们有效地呈现出来。

重要的优化技术将在优化 Flash Platform 的内容指南中进行讨论。该指南中讨论的技术适用于所有 Flash 和 AIR 内容，但是对于开发在移动设备上运行良好的应用程序而言是至关重要的。

- Paul Trani：移动 Flash 开发提示和技巧
- roguish：GPU 测试应用程序 （用于移动设备的 AIR）
- Jonathan Campos：AIR for Android 应用程序的优化技巧
- Charles Schulze：AIR 2.6 游戏开发：包含 iOS

## 应用程序生命周期

当您的应用程序针对其他应用程序丢失焦点时， AIR 会将帧速率降到每秒 4 帧并停止呈现图形。在这种帧速率下，流网络和套接字连接将会中断。如果您的应用程序不使用此类连接，则可以将帧速率调节为甚至更低的值。

在适当的情况下，您应该停止音频播放并删除对 Geolocation 和 Accelerometer 传感器的侦听器。 AIR NativeApplication 对象会调度激活和停用事件。使用这些事件可以管理在活动与背景状态之间的转换。

大多数移动操作系统在终止背景应用程序时都没有警告。通过经常保存应用程序状态，您的应用程序应能够将自己恢复到合理的状态，不论是从背景返回到活动状态还是通过重新启动。

## 信息密度

移动设备屏幕的实际大小小于台式机，但像素密度 （每英寸像素数）却更高。相同字体大小生成的字母在移动设备屏幕上显示的实际上要比在台式机上小。通常必须使用较大的字体才能确保可读性。一般而言， 14 点是易于读取的最小字体大小。

移动设备通常在活动时以及照明条件很差的情况下使用。考虑您实际可以在屏幕上清晰显示多少信息。它可能比您在具有相同像素尺寸的台式机屏幕上显示的内容要少。

同时考虑用户在触摸屏幕时，他们的手指和手会挡住部分显示区。如果用户需要较长时间地触摸某些交互元素，请将这些元素置于屏幕的两侧和底部。

## 文本输入

许多设备使用虚拟键盘进行文本输入。虚拟键盘会隐藏部分屏幕，并且通常使用起来十分麻烦。避免依赖键盘事件 （软键除外）。

考虑实现替代方法以使用输入文本字段。例如，要让用户输入数值，您不需要文本字段。您可以提供两个按钮来增加或减少此值。

## 软键

移动设备包括数目可变的软键。软键是可通过编程方式具有不同功能的按钮。在您的应用程序中遵守关于这些键的平台惯例。

## 屏幕方向更改

可以纵向或横向查看移动内容。考虑您的应用程序如何处理屏幕方向更改。有关更多信息，请参阅舞台方向。

## 屏幕变暗

在播放视频时， AIR 不会自动阻止屏幕变暗。您可以使用 AIR NativeApplication 对象的 systemIdleMode 属性来控制设备是否将进入省电模式。（在某些平台上，必须请求适当的权限才能使此功能起作用。）

## 来电呼叫

用户打电话或接听电话时， AIR 运行时会自动将音频调节到静音。在 Android 上，如果您的应用程序在背景中时播放音频，则应该在应用程序描述符中设置 Android READ_PHONE_STATE 权限。否则， Android 会防止运行时检测电话并自动将音频调节到静音。请参阅 第 65 页的“Android 权限 ”。

## 命中目标

在设计按钮和用户点击的其他用户界面元素时，需要考虑命中目标的大小。这些元素要足够大，以便在触摸屏上使用一个手指即可激活。此外，还要确保目标之间具有足够的空间。在典型的高分辨率电话屏幕上，命中目标区域应约为每一侧 44 像素到 57 像素。

## 应用程序包安装大小

移动设备用于安装应用程序和存储数据的空间通常远远小于台式机。通过删除不使用的资源和库将应用程序包大小降至最低。

在 Android 上，安装应用程序时不会将应用程序包提取到单独的文件中。相反，在访问资源时会将这些资源解压缩到临时存储中。若要将此解压缩的资源存储占用的空间降至最低，请在资源完全加载后关闭文件和 URL 流。

## 文件系统访问

不同的移动操作系统会施加不同的文件系统限制，而且这些限制通常与由桌面操作系统施加的限制有所不同。因此，平台不同，用于保存文件和数据的适当位置也可能各不相同。

文件系统之间存在差异所产生的一个结果是， AIR File 类所提供的常用目录的快捷方式不一定总是可用。下表列出了可在 Android 和 iOS 上使用的快捷方式：

| | Android | iOS |
|---|---|---|
| File.applicationDirectory | 通过 URL 只读 （非本机路径） | 只读 |
| File.applicationStorageDirectory | 可用 | 可用 |
| File.desktopDirectory | SDCard 的根目录 | 不可用 |
| File.documentsDirectory | SDCard 的根目录 | 可用 |
| File.userDirectory | SDCard 的根目录 | 不可用 |
| File.createTempDirectory() | 可用 | 可用 |
| File.createTempFile() | 可用 | 可用 |

## UI 组件

Adobe 正在开发 Flex 框架的移动优化版本。有关详细信息，请参阅 http://labs.adobe.com/technologies/flex/mobile/。

同时还提供适用于移动应用程序的社区组件项目。这些项目包括：

- Keith Peters 的 Minimal Comps
- Derrick Grigg 的 Minimal Comps 可更换皮肤版本
- Todd Anderson 的 as3flobile 组件

# 创建移动设备 AIR 应用程序的工作流程

针对移动设备 （或其他设备）创建 AIR 应用程序的工作流程通常与创建桌面应用程序的工作流程非常相似。主要的工作流程区别出现在打包、调试和安装应用程序时。例如，AIR for Android 应用程序使用本机 Android APK 包格式而不是 AIR 包格式。因此，它们也使用标准 Android 安装和更新机制。

## AIR for Android

以下步骤是开发用于 Android 的 AIR 应用程序的典型步骤：

- 编写 ActionScript 或 MXML 代码。
- 创建 AIR 应用程序描述符文件 （使用 2.5 或更高版本的命名空间）。
- 编译应用程序。
- 将应用程序打包为 Android 包 (.apk)。
- 在设备或 Android 仿真器上安装 AIR 运行时 （如果未安装）。
- 在设备 （或 Android 仿真器）上安装应用程序。
- 在设备上启动应用程序。

可以使用 Adobe Flash Builder、 Adobe Flash Professional CS5 或命令行工具来完成这些步骤。

在 AIR 应用程序完成并打包为 APK 文件之后，您可以将其提交到 Android Market 或通过其他方式进行分发。

## AIR for iOS

以下步骤是开发用于 iOS 的 AIR 应用程序的典型步骤：

- 安装 iTunes。
- 在 Apple iOS Provisioning Portal 上生成必需的开发人员文件和 ID。这些项目包括：
  - 开发人员证书
  - 应用程序 ID
  - 供给配置文件

  在创建供给配置文件时，必须列出您计划要安装应用程序的任意测试设备的 ID。
- 将开发证书和私钥转换为 P12 keystore 文件。
- 编写应用程序 ActionScript 或 MXML 代码。
- 利用 ActionScript 或 MXML 编译器编译应用程序。
- 创建应用程序的图标图片和初始屏幕图片。
- 创建应用程序描述符 （使用 2.6 或更高版本的命名空间）。
- 使用 ADT 对 IPA 文件打包。
- 使用 iTunes 将您的供给配置文件放置在您的测试设备上。
- 在您的 iOS 设备上安装和测试应用程序。您可以使用 iTunes 安装 IPA 文件。

您的 AIR 应用程序一经完成，您可以使用分发证书和供给配置文件重新对其进行打包。然后，就可以将其提交到 Apple 应用程序库。

# 设置移动应用程序属性

对于其他 AIR 应用程序，可以在应用程序描述符文件中设置基本应用程序属性。移动应用程序会忽略某些特定于桌面的属性，例如窗口大小和透明度。移动应用程序还可以使用自己特定于平台的属性。例如，可以在 Android 应用程序中包括 android 元素，在 iOS 应用程序中包括 iPhone 元素。

## 通用设置

某些应用程序描述符设置对所有移动设备应用程序都很重要。

### 所需的 AIR 运行时版本

使用应用程序描述符文件的命名空间指定应用程序所需的 AIR 运行时版本。

在 application 元素中分配的命名空间，很大程度上决定了应用程序可以使用哪些功能。例如，如果应用程序使用 AIR 2.7 命名空间，但用户安装了某个未来版本，那么应用程序仍将参照 AIR 2.7 的行为 （即使在未来版本中已经更改此行为）。只有当您更改命名空间并发布更新时，应用程序才会访问新的行为和功能。不过，安全修补程序不受此规则限制。

如果设备 （如 Android）所使用的运行时不同于应用程序使用的运行时，那么当用户没有所需的 AIR 版本时，系统将提示用户进行安装或升级。在 iPhone 等包含运行时的设备上，不会发生这种情况 （因为需要的版本和应用程序已经在一开始就打包在一起了）。

使用根 application 元素的 xmlns 属性指定命名空间。应该将下列命名空间用于移动应用程序 （具体取决于您的目标移动平台）：

```
iOS 4+ and iPhone 3Gs+ or Android:
<application xmlns="http://ns.adobe.com/air/application/2.7">
iOS only:
<application xmlns="http://ns.adobe.com/air/application/2.0">
```

注：基于 AIR 2.0 SDK，Packager for iPhone SDK 提供针对 iOS 3 设备的支持。有关构建 iOS 3 的 AIR 应用程序的信息，请参阅构建 iPhone 应用程序。AIR 2.6 SDK（和更高版本）在 iPhone 3G、iPhone 4 和 iPad 设备上支持 iOS 4 及更高版本。

**更多帮助主题**

第 158 页的 “application”

## 应用程序标识

对于发布的每个应用程序，以下几个设置应该是唯一的。包括 ID、名称和文件名。

### Android 应用程序 ID

在 Android 上，通过为 AIR ID 加上前缀 “air.” 将其转换为 Android 包名称。这样的话，如果 AIR ID 是 com.example.MyApp，那么 Android 包名称是 air.com.example.MyApp。

```
<id>com.example.MyApp</id>
<name>My Application</name>
<filename>MyApplication</filename>
```

此外，如果该 ID 在 Android 操作系统上不是合法的包名称，它会转换成合法名称。连字符会更改成下划线；若任何 ID 组件以数字开头，会在前面加上大写字母 “A”。例如， ID：3-goats.1-boat，会转换成包名称：air.A3_goats.A1_boat。

注：添加到应用程序 ID 的前缀可以用于标识 Android Market 中的 AIR 应用程序。如果不希望应用程序因前缀而被标识为 AIR 应用程序，您必须对 APK 文件进行解包，更改应用程序 ID，并按照 Opt-out of AIR application analytics for Android 中的说明将其重新打包。

### iOS 应用程序 ID

请将 AIR 应用程序 ID 设置为与您在 Apple iOS Provisioning Portal 中创建的应用程序 ID 匹配。

iOS 应用程序 ID 包含捆绑种子 ID，后面跟着捆绑标识符。捆绑种子 ID 是 Apple 分配给应用程序 ID 的一个字符串，例如 5RM86Z4DJM。捆绑标识符包含一个您选择的反向域样式名称。捆绑标识符可能以星号 (*) 结尾，表示通配符应用程序 ID。如果捆绑标识符以通配符结尾，您可以使用任意合法字符串替换该通配符。

例如：

- 如果您的 Apple 应用程序 ID 为 5RM86Z4DJM.com.example.helloWorld，则您在应用程序描述符中必须使用 com.example.helloWorld。
- 如果您的 Apple 应用程序 ID 为 96LPVWEASL.com.example.* （通配符应用程序 ID），则您可以使用 com.example.helloWorld 或 com.example.anotherApp，或者以 com.example 开头的其他 ID。
- 最后，如果您的 Apple 应用程序 ID 只是捆绑种子 ID 和通配符，如：38JE93KJL.*，则您可以在 AIR 中使用任意应用程序 ID。

指定应用程序 ID 时，请不要包括应用程序 ID 的捆绑种子 ID 部分。

**更多帮助主题**

第 169 页的 “id”

第 165 页的 “filename”

第 176 页的 “name”

## 应用程序版本

在 AIR 2.5 和更高版本中，可以在 versionNumber 元素中指定应用程序版本。不能再使用 version 元素。当为 versionNumber 指定值时，必须使用由点分隔的最多三个数字组成的序列，例如 :“0.1.2”。版本号的每段最多可以具有三个数字。（即，“999.999.999” 是允许的最大版本号）。不必将所有三段都包含在号码中；“1” 和 “1.0” 都是合法的版本号。

也可以使用 versionLabel 元素来指定版本标签。如果添加了版本标签，就会显示版本标签，而不是像在 Android 应用程序信息屏幕等处一样显示版本号。必须为使用 Android Market 分发的应用程序指定版本标签。如果没有在 AIR 应用程序描述符中指定 versionLabel 值，则会将 versionNumber 值分配给 Android 版本标签字段。

```
<!-- AIR 2.5 and later -->
<versionNumber>1.23.7<versionNumber>
<versionLabel>1.23 Beta 7</versionLabel>
```

在 Android 上，AIR versionNumber 转换为 Android 整数 versionCode，转换公式为: a*1000000 + b*1000 + c，其中 a、b 和 c 分别代表 AIR 版本号的组成部分：a.b.c。

#### 更多帮助主题

第 182 页的 “version”

第 183 页的 “versionLabel”

第 183 页的 “versionNumber”

## 主应用程序 SWF

在 initialWindow 元素的 content 子元素中指定主应用程序 SWF 文件。在移动配置文件中定位设备时，必须使用 SWF 文件（不支持基于 HTML 的应用程序）。

```
<initialWindow>
    <content>MyApplication.swf</content>
</initialWindow>
```

必须包括 AIR 包中的文件 （使用 ADT 或 IDE）。如果只是引用应用程序描述符中的名称，不会自动将此文件包括进包中。

## 主屏幕属性

initialWindow 元素的若干子元素控制主应用程序屏幕的初始外观和行为。

- **aspectRatio** — 指定应用程序最初应显示为 portrait 格式 （高度大于宽度）还是 landscape 格式 （高度小于宽度）。

  ```
  <aspectRatio>landscape</aspectRatio>
  ```

- **autoOrients** — 指定舞台是否应随着用户旋转设备或做出与方向相关的其他手势 （如打开或关闭滑动键盘）而自动改变方向。如果设置为 false （默认），则舞台不会随设备改变方向。

  ```
  <autoOrients>true</autoOrients>
  ```

- **fullScreen** — 指定应用程序应占据设备的整个显示屏，还是与标准操作系统窗口样式 （如系统状态栏）共享显示屏。

  ```
  <fullScreen>true</fullScreen>
  ```

- **renderMode** — 指定运行时应使用图形处理单元 (GPU) 还是主要的中心处理单元 (CPU) 渲染应用程序。通常， GPU 渲染可以提高渲染速度，但某些功能 （例如某些混合模式和 PixelBender 筛选器）在 GPU 模式下无法使用。此外，不同设备和不同设备驱动程序的 GPU 功能和限制是不同的。应始终在尽可能多的设备上测试应用程序，特别是使用 GPU 模式时。

  可以将渲染模式设置为 gpu、 cpu 或 auto。默认值是 auto，该设置目前回退到 CPU 模式。请勿对 Flex 应用程序使用 GPU 渲染模式。

  ```
  <renderMode>gpu</renderMode>
  ```

GPU 模式的限制为:

- Flex 框架不支持 GPU 渲染模式。
- 不支持滤镜
- 不支持 PixelBender 混合和填充
- 不支持以下混合模式:图层、 Alpha、擦除、叠加、强光、变亮和变暗
- 不建议在播放视频的应用程序中使用 GPU 渲染模式。
- 在 GPU 渲染模式中,当虚拟键盘打开时,不会正确地将文本字段移动到可见的位置。若要确保文本字段在用户输入文本时可见,请使用舞台和软键盘事件的 softKeyboardRect 属性将文本字段移到可见区域。
- 如果显示对象无法通过 GPU 进行渲染,则根本不会显示。例如,如果将滤镜应用于显示对象,则不会显示该对象。

注: 在 AIR 2.6 以上的版本中,iOS 的 GPU 实现与在更早版本(AIR 2.0 版本)中使用的实现有很大的不同。适用不同的优化注意事项。

### 更多帮助主题

第 161 页的 "aspectRatio"

第 161 页的 "autoOrients"

第 167 页的 "fullScreen"

第 178 页的 "renderMode"

## 支持的配置文件

您可以添加 supportedProfiles 元素,以指定您的应用程序支持哪些设备配置文件。针对移动设备使用 mobileDevice 配置文件。当使用 Adobe Debug Launcher (ADL) 运行应用程序时, ADL 会将列表中的第一个配置文件用作活动配置文件。您也可以在运行 ADL 时使用 -profile 标志,以在支持列表中选择特定的配置文件。如果您的应用程序在所有配置文件下运行,则可以完全忽略 supportedProfiles 元素。在这种情况下, ADL 会将桌面配置文件用作默认的活动配置文件。

若要指定应用程序同时支持移动设备和桌面配置文件,并且您通常要在移动设备配置文件中测试应用程序,请添加以下元素:

```
<supportedProfiles>mobileDevice desktop</supportedProfiles>
```

### 更多帮助主题

第 180 页的 "supportedProfiles"

第 186 页的 " 设备配置文件 "

第 121 页的 "AIR Debug Launcher (ADL)"

## 必需的本机扩展

支持 mobileDevice 配置文件的应用程序可以使用本机扩展。

在应用程序描述符中声明 AIR 应用程序使用的所有本机扩展。下面的例子说明了用于指定两个所需本机扩展的语法:

```
<extensions>
    <extensionID>com.example.extendedFeature</extensionID>
    <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

extensionID 元素的值与扩展描述符文件中的 id 元素的值相同。扩展描述符文件是一个名为 extension.xml 的 XML 文件。已打包在从本机扩展开发人员处接收到的 ANE 文件中。

### 虚拟键盘行为

请将 softKeyboardBehavior 元素设置为 none，以便可以禁用自动平移和调整大小行为，运行时利用这些行为，可以确保在虚拟键盘出现后具有焦点的文本条目字段位于视图中。如果您禁用该自动行为，则在虚拟键盘出现后，确保文本条目区域或其他相关内容可见就是您的应用程序的职责。您可以使用舞台的 softKeyboardRect 属性，结合 SoftKeyboardEvent 来检测何时键盘将打开以及确定键盘遮住的区域。

若要启用该自动行为，请将该元素值设置为 pan：

```
<softKeyboardBehavior>pan</softKeyboardBehavior>
```

由于 pan 为默认值，省略 softKeyboardBehavior 元素也会启用自动键盘行为。

注：当您同时使用 GPU 呈现时，不支持平移行为。

#### 更多帮助主题

第 179 页的 “softKeyboardBehavior”

Stage.softKeyboardRect

SoftKeyboardEvent

## Android 设置

在 Android 平台上，可以使用应用程序描述符的 android 元素将信息添加到 Android 应用程序清单，该清单是 Android 操作系统使用的应用程序属性文件。创建 APK 包时，ADT 会自动生成 Android Manifest.xml 文件。AIR 会将几个属性设置为某些功能运行所需要的值。在 AIR 应用程序描述符的 android 部分设置的任何其他属性都会添加到 Manifest.xml 文件的相应部分。

注：对于大多数 AIR 应用程序，您必须在 android 元素内设置应用程序所需的 Android 权限，但通常不需要设置其他任何属性。

只能设置形式为字符串、整数或布尔值的属性。不支持对应用程序包中的资源的引用进行设置。

### 保留的 Android 清单设置

AIR 会在生成的 Android 清单文档中设置多个清单项目，以确保应用程序和运行时功能正确运行。您不能定义以下设置：

**manifest** 元素

不能设置 manifest 元素的以下属性：

- package
- android:versionCode
- android:versionName
- xmlns:android

**activity** 元素

不能设置主 activity 元素的以下属性：

- android:label
- android:icon

**application** 元素

不能设置 application 元素的以下属性：

- android:theme
- android:name
- android:label
- android:windowSoftInputMode
- android:configChanges
- android:screenOrientation
- android:launchMode

**uses-sdk** 元素

您不能包括 uses-sdk 元素。

## Android 权限

Android 安全模型需要每个应用程序都请求权限，以便使用在安全性或隐私保护方面有作用的功能。打包应用程序时必须指定这些权限，并且不能在运行时进行更改。安装应用程序时， Android 操作系统会通知用户应用程序正在请求哪些权限。如果没有请求某项功能需要的权限，当应用程序访问该功能时， Android 操作系统可能会引发异常，但并非一定会引发异常。运行时会将异常传递给应用程序。在无提示失败情况下，权限失败消息会添加到 Android 系统日志。

在 AIR 中，可以在应用程序描述符的 android 元素内指定 Android 权限。以下格式用于添加权限 （其中 PERMISSION_NAME 是 Android 权限的名称）：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <uses-permission android:name="android.permission.PERMISSION_NAME" />
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

manifest 元素内的 uses-permissions 语句会直接添加到 Android 清单文档。

以下是使用各种 AIR 功能需要的权限：

**ACCESS_COARSE_LOCATION** 允许应用程序通过 Geolocation 类访问 WIFI 和移动电话网络位置数据。

**ACCESS_FINE_LOCATION** 允许应用程序通过 Geolocation 类访问 GPS 数据。

**ACCESS_NETWORK_STATE 和 ACCESS_WIFI_STATE** 允许应用程序通过 NetworkInfo 类访问网络信息。

**CAMERA** 允许应用程序访问摄像头。

注：当您请求使用摄像头功能的权限时， Android 会假设应用程序也需要摄像头。如果摄像头是应用程序的可选功能，应该将 uses-feature 元素添加到摄像头清单中，并将需要的属性设置为 false。请参阅 第 67 页的 “Android 兼容性筛选 ”。

**INTERNET** 允许应用程序提出网络请求。也允许远程调试。

**READ_PHONE_STATE** 允许 AIR 运行时在打电话期间将音频调节到静音。如果您的应用程序在背景中播放音频，则应设置此权限。

**RECORD_AUDIO** 允许应用程序访问麦克风。

**WAKE_LOCK 和 DISABLE_KEYGUARD** 允许应用程序使用 SystemIdleMode 类设置阻止设备休眠。

**WRITE_EXTERNAL_STORAGE** 允许应用程序写入设备上的外部存储卡。

例如，若要为会需要每种权限的应用程序设置该权限，可以将以下内容添加到应用程序描述符：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
                <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
                <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
                <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
                <uses-permission android:name="android.permission.CAMERA" />
                <uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
                <uses-permission android:name="android.permission.INTERNET" />
                <uses-permission android:name="android.permission.READ_PHONE_STATE" />
                <uses-permission android:name="android.permission.RECORD_AUDIO" />
                <uses-permission android:name="android.permission.WAKE_LOCK" />
                <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

**更多帮助主题**

Android 安全性和权限

Android Manifest.permission 类

## Android 自定义 URI 方案

可以使用自定义 URI 方案从网页或本机 Android 应用程序启动 AIR 应用程序。自定义 URI 支持依赖于 Android 清单中指定的方法滤镜，因此在其他平台上不能使用此技术。

若要使用自定义 URI，请将方法滤镜添加到应用程序描述符的 <android> 区块内。必须指定以下示例中的两个 intent-filter 元素。编辑 <data android:scheme="***my-customuri***"/> 语句以反映自定义方案的 URI 字符串。

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <application>
                    <activity>
                        <intent-filter>
                            <action android:name="android.intent.action.MAIN"/>
                            <category android:name="android.intent.category.LAUNCHER"/>
                        </intent-filter>
                        <intent-filter>
                            <action android:name="android.intent.action.VIEW"/>
                            <category android:name="android.intent.category.BROWSABLE"/>
                            <category android:name="android.intent.category.DEFAULT"/>
                            <data android:scheme="my-customuri"/>
                        </intent-filter>
                    </activity>
                </application>
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

方法滤镜会通知 Android 操作系统可以使用您的应用程序执行指定操作。对于自定义 URI 而言，这意味着用户单击了使用该 URI 方案的链接 （浏览器不知道该如何处理）。

通过自定义 URI 调用应用程序时， NativeApplication 对象会调度一个 invoke 事件。此链接的 URL （包括查询参数）放在 InvokeEvent 对象的 arguments 数组中。您可以使用任意多个方法滤镜。

注：StageWebView 实例中的链接无法打开使用自定义 URI 方案的 URL。

**更多帮助主题**

Android 方法滤镜

Android 操作和类别

## Android 兼容性筛选

Android 操作系统使用应用程序清单文件中的大量元素来确定应用程序与指定设备是否兼容。可以选择将此信息添加到清单中。如果不包括这些元素，则可以将您的应用程序安装在任何 Android 设备上。但是，它可能不会在任何 Android 设备上都正常运行。例如，摄像头应用程序在没有摄像头的电话上将没有什么用处。

可以用于过滤的 Android 清单标签包括：

- supports-screens
- uses-configuration
- uses-feature

摄像头应用程序

如果为应用程序请求摄像头权限， Android 会假设应用程序需要所有可用的摄像头功能，包括自动聚焦和闪光。如果应用程序不需要所有摄像头功能，或者如果摄像头是可选功能，则应对摄像头的各个 uses-feature 元素进行设置以指示这些功能是可选的。否则，使用缺少某项功能或根本没有摄像头的设备的用户将无法找到 Android Market 上的应用程序。

以下示例说明了如何为摄像头请求权限，以及如何将所有摄像头功能设置为可选：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <uses-permission android:name="android.permission.CAMERA" />
                <uses-feature android:name="android.hardware.camera" android:required="false"/>
                <uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
                <uses-feature android:name="android.hardware.camera.flash" android:required="false"/>
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

录音应用程序

如果您请求录音权限，则 Android 还会假设该应用程序需要麦克风。如果录音是您的应用程序的一个可选功能，则可以添加 uses-feature 标签以指定不需要麦克风。否则，使用不带麦克风设备的用户在 Android Market 上将找不到您的应用程序。

下面的例子说明了如何请求使用麦克风的权限，同时仍然使麦克风硬件处于可选状态：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <uses-permission android:name="android.permission.RECORD_AUDIO" />
                <uses-feature android:name="android.hardware.microphone" android:required="false"/>
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

**更多帮助主题**

Android 开发人员：Android 兼容性

Android 开发人员：Android 功能名称常量

## 安装位置

通过将 Android manifest 元素的 installLocation 属性设为 auto 或 preferExternal，您可以允许将应用程序安装或移动到外部存储卡上：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest android:installLocation="preferExternal"/>
        ]]>
    </manifestAdditions>
</android>
```

Android 操作系统不保证您的应用程序会安装到外部内存上。用户也可以使用系统设置应用程序将应用程序在内部与外部内存之间进行移动。

即使安装到外部内存，应用程序缓存和用户数据 （如应用程序存储目录的内容、共享对象和临时文件）仍会存储在内部内存上。若要避免使用太多的内部内存，对于要保存到应用程序存储目录的数据应有所选择。应使用 File.userDirectory 或 File.documentsDirectory 位置 （这两者都会映射到 Android 上 SD 卡的根目录）将大量数据保存到 SDCard 上。

## 在 StageWebView 对象中启用 Flash Player 和其他插件

在 Android 3.0 以上的版本中，应用程序必须在 Android 应用程序元素中启用硬件加速，才能在 StageWebView 对象中显示插件内容。要启用插件渲染，可将 application 元素的 android:hardwareAccelerated 属性设置为 true：

```
<android>
    <manifestAdditions>
        <![CDATA[
            <manifest>
                <application android:hardwareAccelerated="true"/>
            </manifest>
        ]]>
    </manifestAdditions>
</android>
```

AIR 会验证包含在应用程序描述符 Android 部分中的元素和属性。默认设置下， AIR 会依照 Android 2.2 SDK 进行验证。由于在 Android 3.0 中添加了 android:hardwareAccelerated 属性，您必须设置 AIR ADT 实用程序以依照 Android 3.0 SDK （或更新版本）进行验证。

首先下载 Android 3 SDK：Android 开发人员：安装 SDK 。在打包过程中，将 -platformsdk 选项设置为包含相应 Android SDK 的路径 （将路径设置为包含 Android tools 文件夹的目录）。例如：

```
adt -package
    -target apk
    -storetype pkcs12 -keystore cert.p12 -storepass foo
    myApp.apk
    myApp-app.xml
    -platformsdk c:\androidSDK
    myApp.swf
    other.files
```

也可将 AIR_ANDROID_SDK_HOME 环境变量设置为包含 Android SDK 的路径。由于 Flash Builder 和 Flash Professional 不允许您在打包时添加额外参数，因此设置此环境变量可让您在自己使用的 AIR SDK 释放时使用不可用的 Android SDK 设置。请参阅 第 141 页的 “ADT 环境变量 ”。

### 颜色深度

**AIR 3+**

在 AIR 3 和更高版本中，运行时将显示设置为呈现 32 位颜色。在早期版本的 AIR 中，运行时使用 16 位颜色。您可通过使用应用程序描述符中的 <colorDepth> 元素，指示运行时使用 16 位颜色：

```
<android>
    <colorDepth>16bit</colorDepth>
    <manifestAdditions>...</manifestAdditions>
</android>
```

使用 16 位颜色深度可提高渲染性能，但是会牺牲颜色保真度。

## iOS 设置

仅应用于 iOS 设备的设置会被放置到应用程序描述符的 <iPhone> 元素中。 iPhone 元素可以包含 InfoAdditions 元素和 requestedDisplayResolution 元素作为子元素。

利用 InfoAdditions 元素，您可以指定要添加到应用程序的 Info.plist 设置文件的键值对。例如，下面这些值设置应用程序的状态栏样式，并声明应用程序不要求永久的 Wi-Fi 访问。

```
<InfoAdditions>
    <![CDATA[
        <key>UIStatusBarStyle</key>
        <string>UIStatusBarStyleBlackOpaque</string>
        <key>UIRequiresPersistentWiFi</key>
        <string>NO</string>
    ]]>
</InfoAdditions>
```

InfoAdditions 设置括在 CDATA 标记中。

有关 Info.plist 设置的更多详细信息，请参阅 Apple 开发人员文档。

### 保留的 iOS InfoAdditions 设置

AIR 会在生成的 Info.plist 文件中设置多个条目，以确保应用程序和运行时功能正确运行。您不能定义以下设置：

| | |
|---|---|
| CFBundleDisplayName | CTAutoOrients |
| CFBundleExecutable | CTInitialWindowTitle |
| CFBundleIconFiles | CTInitialWindowVisible |
| CFBundleIdentifier | DTPlatformName |
| CFBundleInfoDictionaryVersion | DTSDKName |
| CFBundlePackageType | MinimumOSVersion |
| CFBundleResourceSpecification | NSMainNibFile |
| CFBundleShortVersionString | UIInterfaceOrientation |
| CFBundleSupportedPlatforms | UIStatusBarHidden |
| CFBundleVersion | UISupportedInterfaceOrientations |

## 支持不同的 iOS 设备型号

为了支持 iPad，请为 InfoAdditions 元素中的 UIDeviceFamily 添加正确的键值设置。UIDeviceFamily 设置是一个字符串数组。每个字符串都定义受支持的设备。<string>1</string> 设置定义对 iPhone 和 iPod Touch 的支持。<string>2</string> 设置定义对 iPad 的支持。如果仅指定其中一个字符串，则仅支持该设备系列。例如，下面的设置限制对 iPad 的支持：

```
<key>UIDeviceFamily</key>
    <array>
        <string>2</string>
    </array>>
```

下面的设置支持两个设备系列 （iPhone/iPod Touch 和 iPad）：

```
<key>UIDeviceFamily</key>
<array>
    <string>1</string>
    <string>2</string>
</array>
```

## 高分辨率显示器

利用 requestedDisplayResolution 元素，可以指定在具有高分辨率屏幕的 iOS 设备上，应用程序是应该使用 standard 分辨率模式还是应该使用 high 分辨率模式。

```
<requestedDisplayResolution>high</requestedDisplayResolution>
```

在高分辨率模式中，您可以在高分辨率显示屏上逐个处理每个像素。在标准模式中，设备屏幕将以标准分辨率显示应用程序。在该模式中绘制单个像素会在高分辨率屏幕上设置四个像素的颜色。

默认设置为 standard。请注意， requestedDisplayResolution 元素是 iPhone 元素 （而不是 InfoAdditions 元素）的子元素。

### 更多帮助主题

第 179 页的 “requestedDisplayResolution”

Renaun Erickson：使用 AIR 2.6 开发 Retina 和非 Retina iOS 屏幕

## iOS 自定义 URI 方案

您可以注册自定义 URI 方案，以便允许您的应用程序由网页中的链接或设备上的其他本机应用程序进行调用。若要注册 URI 方案，请将 CFBundleURLTypes 键添加到 InfoAdditions 元素中。下面的示例注册了一个名为 com.example.app 的 URI 方案，从而允许应用程序由 example://foo 形式的 URL 进行调用。

```
<key>CFBundleURLTypes</key>
<array>
    <dict>
         <key>CFBundleURLSchemes</key>
        <array>
            <string>example</string>
        </array>
        <key>CFBundleURLName</key>
        <string>com.example.app</string>
    </dict>
</array>
```

通过自定义 URI 调用应用程序时， NativeApplication 对象会调度一个 invoke 事件。此链接的 URL （包括查询参数）放在 InvokeEvent 对象的 arguments 数组中。您可以使用任意多个自定义 URI 方案。

注：StageWebView 实例中的链接无法打开使用自定义 URI 方案的 URL。

注：如果另一个应用程序已注册某个方案，则您的应用程序不能将其替换成为该 URI 方案注册的应用程序。

### iOS 兼容性筛选

如果您的应用程序只能在具有特定硬件或软件功能的设备上使用，则请将条目添加到 InfoAdditions 元素的 UIRequiredDeviceCapabilities 数组中。例如，以下条目表示应用程序需要静态摄像头和麦克风：

```
<key>UIRequiredDeviceCapabilities</key>
<array>
    <string>microphone</string>
    <string>still-camera</string>
</array>
```

如果设备缺少相应的功能，则无法安装该应用程序。与 AIR 应用程序相关的功能设置包括：

| | |
|---|---|
| telephony | camera-flash |
| wifi | video-camera |
| sms | accelerometer |
| still-camera | location-services |
| auto-focus-camera | gps |
| front-facing-camera | microphone |

AIR 2.6 以上的版本自动将 armv7 和 opengles-2 添加到必需功能的列表中。

注：您无需为了让应用程序使用这些功能而将其添加到应用程序描述符中。仅需使用 UIRequiredDeviceCapabilities 设置即可阻止用户在无法正常使用的设备上安装应用程序。

### 退出而不是暂停

如果用户离开 AIR 应用程序，该应用程序会进入后台并暂停。如果要让应用程序彻底退出而不是暂停，请将 UIApplicationExitsOnSuspend 属性设置为 YES：

```
<key>UIApplicationExitsOnSuspend</key>
<true/>
```

## 应用程序图标

下表列出了每个移动设备平台使用的图标尺寸：

| 图标尺寸 | 平台 |
|---|---|
| 29x29 | iOS |
| 36x36 | Android |
| 48x48 | Android, iOS |
| 57x57 | iOS |
| 72x72 | Android, iOS |
| 114x114 | iOS |
| 512x512 | iOS |

指定应用程序描述符文件的图标元素中图标文件的路径：

```
<icon>
    <image36x36>assets/icon36.png</image36x36>
    <image48x48>assets/icon48.png</image48x48>
    <image72x72>assets/icon72.png</image72x72>
</icon>
```

如果未提供指定尺寸的图标，则使用第二大尺寸并缩放至适合的大小。

#### Android 上的图标

在 Android 上，应用程序描述符中指定的图标会用作应用程序 Launcher 图标。应用程序 Launcher 图标应作为一组 36x36 像素、 48x48 像素和 72x72 像素 PNG 图像提供。这些图标尺寸分别用于低密度、中密度和高密度屏幕。

#### iOS 上的图标

在应用程序描述符中定义的图标用于 iOS 应用程序的以下位置：

- 29 x 29 像素图标 — iPhone 和 iPod 触摸屏上的 Spotlight 搜索结果使用此图标。
- 48 x 48 像素图标 — iPad 上的 Spotlight 搜索结果使用此图标。
- 57 x 57 像素图标 — iPhone 和 iPod Touch 主屏幕显示此图标。
- 72 x 72 像素图标 （可选） — iPad 主屏幕显示此图标。
- 114 x 114 像素图标 — iPhone 和 iPod Touch 主屏幕图标，用于 960 x 640 像素显示。
- 512 × 512 像素图标 — iTunes 显示此图标。当您将最终应用程序提交给 Apple 应用程序库时， 512 像素 PNG 文件仅用于测试应用程序的开发版本，须单独以 JPG 文件格式提交 512 图像。它不包含在 IPA 中。

iOS 为图标添加了眩光效果。您无需对源图像应用这种效果。要删除此默认眩光效果，请将以下内容添加到应用程序描述符文件中的 InfoAdditions 元素：

```
<InfoAdditions>
    <![CDATA[
        <key>UIPrerenderedIcon</key>
        <true/>
    ]]>
</InfoAdditions>
```

注：在 iOS 上，已将短语 “Adobe AIR for iOS” 作为元数据添加到应用程序图标，这样 Adobe 可以跟踪 Apple iOS 应用程序库中可用 AIR 应用程序的数量。如果不希望应用程序因此图标元数据而被标识为 AIR 应用程序，您必须对 IPA 文件进行解包，删除图标元数据，并按照 退出 AIR 应用程序分析 （适用于 iOS） 中的说明对其重新打包。

#### 更多帮助主题

第 168 页的 “icon”

第 169 页的 “imageNxN”

Android 开发人员：图标设计指南

iOS 人机界面指南：自定义图标和图像创建指南

### iOS 启动图像

除应用程序图标外，必须至少还提供一个名为 “Default.png” 的启动图像。或者，您可以为不同的启动方向、不同的分辨率以及不同的设备单独提供启动图像。您还可以添加不同的启动图像，以便当通过 URL 调用您的应用程序时使用。

启动图像文件未在应用程序描述符中引用，必须放置在应用程序的根目录中。（请勿 将该文件放在子目录中。）

#### 文件命名方案

根据以下方案对图像命名：

```
basename + urischeme + orientation + scale + device + .png
```

文件名的 basename 部分可以是 Default （D 大写），也可以是使用应用程序描述符的 InfoAdditions 元素中的 UILaunchImageFile 键指定的名称。

urischeme 部分是用于标识 URI 方案的字符串。例如，如果可以通过链接 （如 example://foo）调用您的应用程序，则 example 可用作启动图像文件名的方案部分。

orientation 部分可以为以下任意值之一，这些值用于指示当应用程序启动时设备所处的方向：

- -Portrait
- -PortraitUpsideDown
- -Landscape
- -LandscapeLeft
- -LandscapeRight

对于用于高分辨率显示屏的启动图像， scale 部分为 @2x。（对于用于标准分辨率显示屏的图像，请忽略整个 scale 部分。）

device 部分可以是 ~ipad，也可以是 ~iphone （包含 iPhone 和 iPod Touch）。

对于 iPhone，只能添加纵向高宽比图像。对于标准分辨率设备，请使用 320x480 像素的图像，对于高分辨率设备，请使用 640x960 像素的图像。

对于 iPad，横向 (1024x748) 和纵向 (768x1004) 高宽比图像均可添加。（目前， iPad 均未配备高分辨率屏幕。）

### 示例

下表列出了可以为一个假想的应用程序添加的启动图像集示例，该应用程序支持最广泛的设备和方向，可通过 URL 使用 example:// 方案启动：

| 文件名 | 图像大小 | 用法 |
|---|---|---|
| Default.png | 320 x 480 | iPhone，标准分辨率 |
| Default@2x.png | 640 x 960 | iPhone，高分辨率 |
| Default-Portrait.png | 768 x 1004 | iPad，纵向 |
| Default-PortraitUpsideDown.png | 768 x 1004 | iPad，倒置纵向 |
| Default-Landscape.png | 1024 x 748 | iPad，左横向 |
| Default-LandscapeRight.png | 1024 x 748 | iPad，右横向 |
| Default-example.png | 320 x 480 | example:// 标准 iPhone 上的 URL |
| Default-example@2x.png | 640 x 960 | example:// 高分辨率 iPhone 上的 URL |
| Default-example~ipad.png | 768 x 1004 | example:// 纵向 iPad 上的 URL |
| Default-example-Landscape.png | 1024 x 748 | example:// 横向 iPad 上的 URL |

本示例仅说明了一种方法。例如，可以为 iPad 使用 Default.png 图像，使用 Default~iphone.png 和 Default@2x~iphone.png 为 iPhone 和 iPod 指定特定启动图像。

另请参见

iOS Application Programming Guide: Application Launch Images

图片指南

您可以为启动图像创建任何图片，只要尺寸正确即可。但是，通常最好将图像与应用程序的初始状态相匹配。您可以捕获应用程序启动屏幕的屏幕快照，从而创建启动图像：

1 在 iOS 设备上打开应用程序。用户界面的第一个屏幕出现后，按住 " 主屏幕 " 按钮 (位于屏幕下方)。在按住 " 主屏幕 " 按钮的同时，按下 " 睡眠 / 唤醒 " 按钮 (位于设备顶部)。这会捕获一张屏幕快照并将其发送到摄像头卷。

2 通过从 iPhoto 或其他照片传输应用程序传输照片将此图像传输到您的开发计算机。

如果您的应用程序已本地化为多种语言，请勿在启动图像中添加文本。启动图像是静态的，文本可能与其他语言不匹配。

另请参见

iOS Human Interface Guidelines: Launch images

## 忽略的设置

移动设备上的应用程序会忽略应用于本机窗口或桌面操作系统功能的应用程序设置。忽略的设置包括：

- allowBrowserInvocation
- customUpdateUI
- fileTypes
- height
- installFolder
- maximizable
- maxSize
- minimizable
- minSize
- programMenuFolder
- resizable
- systemChrome
- title
- transparent
- visible
- width
- x
- y

# 打包移动 AIR 应用程序

使用 ADT -package 命令可以为预期在移动设备上使用的 AIR 应用程序创建应用程序包。-target 参数指定为哪个移动平台创建了该包。

### Android 包

Android 上的 AIR 应用程序使用 Android 应用程序包格式 (APK)，而不是 AIR 包格式。

ADT 使用 APK 目标类型生成的包，其格式为可提交到 Android Market 的格式。 Android Market 要求提交的应用程序必须满足接受条件。在创建最终软件包之前，您应查看最新的要求。请参阅 Android 开发人员：在 Market 上发布。

可以使用标准 AIR 代码签名证书来对应用程序进行签名；但是，若要将应用程序提交到 Android Market，证书必须符合 Market 规则，这些规则要求证书的有效期至少到 2033 年。可以使用 ADT -certificate 命令创建此类证书。

Developer Serge Jespers 创建了 Package Assistant Pro，该 AIR 应用程序可以协助为 Android 以及桌面打包 AIR 应用程序。可以在 http://www.webkitchen.be/package-assistant-pro/ 下载此应用程序。

若要将应用程序提交到替代 Market，而该 Market 不允许您的应用程序请求从 Google Market 下载 AIR，则可使用 ADT 的 -airDownloadURL 参数指定替代下载 URL。如果用户没有启动应用程序所需的 AIR 运行时版本，则会直接将其引导至指定的 URL。有关更多信息，请参阅 第 126 页的 “ADT package 命令 ”。

在 AIR 3 或更高版本中，可将 AIR 运行时的捕获副本和您的 Android 应用程序捆绑到一起。您的应用程序将独占使用 AIR 的此捕获版本。用户无需单独安装 AIR 运行时。

**iOS 包**

iOS 上的 AIR 应用程序使用 iOS 软件包格式 (IPA)，而不是本机 AIR 格式。

ADT 使用 ipa-app-store 目标类型、正确的代码签名证书和供给配置文件生成的包，其格式为可提交到 Apple 应用程序库的格式。使用 ipa-ad-hoc 目标类型可以对某个应用程序进行打包，以进行临时分发。

必须使用 Apple 颁发的正确的开发人员证书对应用程序进行签名。创建测试版所用的证书与应用程序提交前进行最终打包所用的证书不同。

有关如何使用 Ant 打包 iOS 应用程序的示例，请参阅 Piotr Walczyszyn: 使用 ADT 命令和 ANT 脚本打包 iOS 设备的 AIR 应用程序

## 使用 ADT 打包

AIR SDK 版本 2.6 和更高版本支持 iOS 和 Android 打包。打包前必须先编译所有的 ActionScript、 MXML 和任何扩展代码。还必须有代码签名证书。

有关 ADT 命令和选项的详细参考，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。

### Android APK 包

**创建 APK 包**

若要创建 APK 包，请使用 ADT 包命令，同时针对发行版将目标类型设为 apk，针对调试版本设为 apk-debug，或针对发行模式版本设为 apk-emulator，以便在仿真器上运行。

```
adt -package
    -target apk
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
```

请在一行内键入完整的命令；上面示例中出现换行符仅是为了阅读更加方便。另外，此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 “ 路径环境变量 ”。）

必须从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、 myApp.swf 和图标目录。

当运行如上所示的命令时， ADT 会提示输入 keystore 密码。（键入的密码字符不会显示；只需在键入结束后按 Enter。）

### 为使用本机扩展的应用程序创建 APK 包

若要为使用本机扩展的应用程序创建 APK 包，除了标准打包选项外，还请添加 -extdir 选项。此选项指定包含应用程序所使用的 ANE 文件的目录。例如：

```
adt -package
    -target apk
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    -extdir extensionsDir
    myApp.swf icons
```

### 创建包含自有 AIR 运行时版本的 APK 包

若要创建既包含应用程序又包含 AIR 运行时的捕获版本的 APK 包，请使用 apk-captive-runtime 目标。此选项指定包含应用程序所使用的 ANE 文件的目录。例如：

```
adt -package
    -target apk-captive-runtime
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
```

如果将此运行时与您的应用程序捆绑到一起，您的应用程序将独占使用此捕获运行时。

与捕获运行时打包在一起的优点有：

- 生成自包含应用程序
- 安装程序与运行时更新隔离
- 企业可以认证特定应用程序和运行时组合
- 无需单独下载和安装运行时

另一方面，存在以下一些缺点：

- 当 Adobe 发布安全修补程序时，不会自动向用户提供关键安全修补程序
- 需要更多的应用程序 RAM 空间

注：如果您捆绑运行时，ADT 将向您的应用程序添加 INTERNET 和 BROADCAST_STICKY 权限。这些权限是 AIR 运行时的必需权限。

### 创建调试 APK 包

若要创建可以与调试器一起使用的应用程序版本，请使用 apk-debug 作为目标，并指定连接选项：

```
adt -package
    -target apk-debug
    -connect 192.168.43.45
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
```

-connect 标志会告知设备上的 AIR 运行时通过网络连接到远程调试器的位置。若要通过 USB 进行调试，您必须改为指定 -listen 标志，以指定用于调试连接的 TCP 端口：

```
adt -package
    -target apk-debug
    -listen 7936
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
```

若要使大部分调试功能正常工作，还必须在启用调试的情况下编译应用程序 SWF 和 SWC。有关 -connect 和 -listen 标志的完整说明，请参阅第 137 页的 “ 调试器连接选项 ”。

在 Android 上，应用程序还必须有访问 Internet 的权限，以便通过网络连接到运行调试器的计算机。请参阅 第 65 页的 “Android 权限 ”。

**创建 APK 包以便在 Android 模拟器上使用**

您可以在 Android 模拟器上使用调试 APK 包，而不是释放模式包。若要创建一个释放模式 APK 包以便在模拟器上使用，请使用 ADT package 命令，将目标类型设置为 apk-emulator：

```
adt -package -target apk-emulator -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp-app.xml
myApp.swf icons
```

此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 “ 路径环境变量 ”。）

**从 AIR 或 AIRI 文件创建 APK 包**

可以从现有的 AIR 或 AIRI 文件直接创建 APK 包：

```
adt -target apk -storetype pkcs12 -keystore ../codesign.p12 myApp.apk myApp.air
```

AIR 文件必须使用应用程序描述符文件中的 AIR 2.5 （或更高版本）命名空间。

## iOS 包

在 iOS 上， ADT 将 SWF 文件字节代码和其他源文件转换为本机 iOS 应用程序。

1 打开命令外壳或终端并导航到您的 iPhone 应用程序的项目文件夹。

2 然后，使用 ADT 工具创建 IPA 文件，语法如下：

```
adt -package
    -target [ipa-test | ipa-debug | ipa-app-store | ipa-ad-hoc]
    -provisioning-profile PROFILE_PATH
    SIGNING_OPTIONS
    TARGET_IPA_FILE
    APP_DESCRIPTOR
    SOURCE_FILES
    -extdir extension-directory
```

更改引用 adt 以包括 adt 应用程序的完整路径。 adt 应用程序安装在 AIR SDK 的 bin 子目录中。

选择对应于您要创建的 iPhone 应用程序类型的 -target 选项：

- -target ipa-test — 选择此选项可快速编译要在开发人员 iPhone 上进行测试的应用程序版本。
- -target ipa-debug — 选择此选项可编译要在开发人员 iPhone 上测试的应用程序调试版本。通过此选项，您可以使用调试会话从 iPhone 应用程序接收 trace() 输出。

  您可以包含以下 -connect 选项 (CONNECT_OPTIONS) 之一，以指定运行该调试器的开发计算机的 IP 地址：

  - -connect — 应用程序将尝试连接到用于编译该应用程序的开发计算机上的调试会话。
  - -connect IP_ADDRESS — 应用程序将尝试连接到具有指定 IP 地址的计算机上的调试会话。例如：

    ```
    -target ipa-debug -connect 192.0.32.10
    ```

- -connect HOST_NAME — 应用程序将尝试连接到具有指定主机名称的计算机上的调试会话。例如：

```
-target ipa-debug -connect bobroberts-mac.example.com
```

-connect 选项是可选项。如果不指定该选项，则生成的调试应用程序不会尝试连接到托管调试器。

如果调试连接尝试失败，应用程序会显示一个对话框，要求用户输入调试主机的 IP 地址。如果设备未连接到 wifi，连接尝试可能失败。如果设备已连接，但不在调试主机的防火墙之后，也可能失败。

有关更多信息，请参阅第 80 页的 " 调试移动 AIR 应用程序 "。

- -target ipa-ad-hoc — 选择此选项可创建用于临时部署的应用程序。请参见 Apple iPhone 开发人员中心
- -target ipa-app-store — 选择此选项可创建用于部署到 Apple 应用程序库的 IPA 文件的最终版本。

用应用程序的供给配置文件的路径替换 PROFILE_PATH。有关供给配置文件的更多信息，请参阅 第 56 页的 "iOS 设置 "。

替换 SIGNING_OPTIONS 以引用 iPhone 开发人员证书和密码。应使用以下语法：

```
-storetype pkcs12 -keystore P12_FILE_PATH -storepass PASSWORD
```

使用 P12 证书文件的路径替换 P12_FILE_PATH。使用证书密码替换 PASSWORD。（请参见以下示例。）有关 P12 证书文件的更多信息，请参阅第 149 页的 " 将开发人员证书转换为 P12 keystore 文件 "。

替换 APP_DESCRIPTOR 以引用应用程序描述符文件。

替换 SOURCE_FILES 以引用项目 （后面跟有任何其他要包含的资源）的主 SWF 文件。包括在 Flash CS5 中的应用程序设置对话框中或在自定义应用程序描述符文件中定义的所有图标文件的路径。此外，还要添加初始屏幕图片文件 Default.png。

使用 -extdir ***extension-directory*** 选项可指定包含应用程序使用的 ANE 文件 （本机扩展）的目录。如果应用程序不使用本机扩展，请勿包含此选项。

重要说明：请勿在您的应用程序目录中创建名为 Resources 的子目录。运行时将自动创建具有此名称的文件夹以符合 IPA 包的结构要求。如果您自己创建 "Resources" 文件夹将导致致命冲突。

### 创建 iOS 包以便调试

若要创建一个 iOS 包以便安装在测试设备上，请使用 ADT package 命令，将目标类型设置为 ios-debug。在运行此命令前，必须已经从 Apple 获取一个开发代码签名证书和供给配置文件。

```
adt -package
    -target ipa-debug
    -storetype pkcs12 -keystore ../AppleDevelopment.p12
    -provisioning-profile AppleDevelopment.mobileprofile
    -connect 192.168.0.12
    myApp.ipa
    myApp-app.xml
    myApp.swf icons Default.png
```

请在一行内键入完整的命令；上面示例中出现换行符仅是为了阅读更加方便。另外，此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 " 路径环境变量 "。）

必须从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、myApp.swf、图标目录和 Default.png 文件。

必须使用由 Apple 颁发的正确分发证书对应用程序进行签名；不得使用其他代码签名证书。

当指定 -connect 选项时，应用程序将尝试利用在指定 IP 或主机名上运行的 Flash 调试器 (FDB) 来启动调试会话。请参阅第 81 页的 " 连接到 Flash 调试器 " 以了解更多信息。

### 创建 iOS 包以便提交到 Apple 应用程序库

若要创建一个 iOS 包以便提交到 Apple 应用程序库，请使用 ADT package 命令，将目标类型设置为 ios-app-store。在运行此命令前，您必须已经从 Apple 获取一个分发代码签名证书和供给配置文件。

```
adt -package
    -target ipa-app-store
    -storetype pkcs12 -keystore ../AppleDistribution.p12
    -provisioning-profile AppleDistribution.mobileprofile
    myApp.ipa
    myApp-app.xml
    myApp.swf icons Default.png
```

请在一行内键入完整的命令；上面示例中出现换行符仅是为了阅读更加方便。另外，此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 " 路径环境变量 "。）

必须从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、myApp.swf、图标目录和 Default.png 文件。

必须使用由 Apple 颁发的正确分发证书对应用程序进行签名；不得使用其他代码签名证书。

重要说明：Apple 要求您使用 Application Loader 程序，以便将应用程序上载到应用程序库。 Apple 仅针对 Mac OS X 发布了 Application Loader。因此，当您使用 Windows 计算机为 iPhone 开发 AIR 应用程序时，必须具有访问运行 OS X （版本 10.5.3 或更高版本）的计算机的权限，以便将应用程序提交到应用程序库。您可以从 Apple iOS 开发人员中心获取 Application Loader 程序。

### 创建 **iOS** 包以便进行临时分发

若要创建 iOS 包以便进行临时分发，请使用 ADT package 命令，将目标类型设置为 ios-ad-hoc。在运行此命令前，您必须已经从 Apple 获取适当的临时分发代码签名证书和供给配置文件。

```
adt -package
    -target ipa-ad-hoc
    -storetype pkcs12 -keystore ../AppleDistribution.p12
    -provisioning-profile AppleDistribution.mobileprofile
    myApp.ipa
    myApp-app.xml
    myApp.swf icons Default.png
```

请在一行内键入完整的命令；上面示例中出现换行符仅是为了阅读更加方便。另外，此示例假设指向 ADT 工具的路径存在于命令行 shell 的路径定义中。（有关帮助信息，请参阅第 237 页的 " 路径环境变量 "。）

必须从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、myApp.swf、图标目录和 Default.png 文件。

必须使用由 Apple 颁发的正确分发证书对应用程序进行签名；不得使用其他代码签名证书。

### 为使用本机扩展的应用程序创建 **iOS** 包

要为使用本机扩展的应用程序创建 iOS 包，请使用具有 -extdir 选项的 ADT 包命令。使用与目标 （ipa-app-store、 ipa-debug、ipa-ad-hoc、 ipa-test）相适应的 ADT 命令。例如：

```
adt -package
    -target ipa-ad-hoc
    -storetype pkcs12 -keystore ../AppleDistribution.p12
    -provisioning-profile AppleDistribution.mobileprofile
    myApp.ipa
    myApp-app.xml
    -extdir extensionsDir
    myApp.swf icons Default.png
```

请在一行内键入完整的命令；上面示例中出现换行符仅是为了阅读更加方便。

关于本机扩展，本示例假设名为 extensionsDir 的目录位于运行命令的目录中。extensionsDir 目录包含应用程序所使用的 ANE 文件。

# 调试移动 AIR 应用程序

可以用多种方法调试移动 AIR 应用程序。发现应用程序逻辑问题的最简单方法是使用 ADL 在开发计算机上进行调试。也可以在设备上安装应用程序，并使用在台式计算机上运行的 Flash 调试器进行远程调试。

## 使用 ADL 的设备模拟

用来测试和调试大多数移动应用程序功能的一种最快、最简单的方法是：使用 Adobe Debug Launcher (ADL) 实用程序在开发计算机上运行您的应用程序。ADL 使用应用程序描述符中的 supportedProfiles 元素来决定要使用的配置文件。如果列出多个配置文件，则 ADL 会使用列表中的第一个配置文件。您也可以使用 ADL 的 -profile 参数来选择 supportedProfiles 列表中的某个其他配置文件。（如果应用程序描述符中不包括 supportedProfiles 元素，则可以针对 -profile 参数指定任何配置文件。）例如，使用以下命令启动应用程序以模拟移动设备配置文件：

```
adl -profile mobileDevice myApp-app.xml
```

这样在桌面上模拟移动配置文件时，应用程序会在与目标移动设备更为相似的环境中运行。不属于移动配置文件一部分的 ActionScript API 不可用。不过， ADL 并不区分不同移动设备的功能。例如，您可以将模拟软键的按键方式发送到您的应用程序，即便您实际的目标设备不利用软键。

ADL 支持通过菜单命令模拟设备方向变化和软键输入。在移动设备配置文件中运行 ADL 时， ADL 会显示允许您进入设备旋转或软键输入的菜单 （在应用程序窗口或桌面菜单栏中）。

**软键输入**

ADL 模拟移动设备上的 “ 后退 ”、 “ 菜单 ” 和 “ 搜索 ” 软键按钮。当使用移动配置文件启动 ADL 时，您可以使用所显示的菜单将这些键发送到模拟设备上。

**设备旋转**

当使用移动配置文件启动 ADL 时， ADL 可让您通过所显示的菜单模拟设备旋转。您可以将模拟设备旋转到右侧或左侧。

旋转模拟只会影响支持自动定向的应用程序。您可以通过在应用程序描述符中将 autoOrients 元素设为 true 来启用此功能。

**屏幕大小**

您可以通过设置 ADL -screensize 参数在不同尺寸的屏幕上测试您的应用程序。您可以将代码传递给预定义屏幕类型之一，或传递给包含四个表示正常屏幕和最大化屏幕的像素尺寸值的字符串。例如，以下命令将打开 ADL 以模拟在 Motorola Droid 上使用的屏幕：

```
adl -screensize 480x816:480x854 myApp-app.xml
```

欲访问预定义屏幕类型的列表，请参阅 第 121 页的 “ADL 用法 ”。

**限制**

ADL 无法模拟在桌面配置文件上不受支持的某些 API。不能模拟的 API 包括：

- Accelerometer
- cacheAsBitmapMatrix
- CameraRoll
- CameraUI
- Geolocation
- 不支持这些功能的桌面操作系统上的多点触控和手势
- SystemIdleMode

如果您的应用程序使用这些类，则应在实际设备或仿真器上测试这些功能。

同样，当在桌面上的 ADL 下运行时有一些 API 可以工作，但它们并非在所有类型的移动设备上都可以工作。这些项目包括：

- Speex 和 AAC 视频编解码器
- 辅助功能和屏幕阅读支持
- RTMPE
- 加载包含 ActionScript 字节代码的 SWF 文件
- PixelBender 着色器

请确保在目标设备上对使用这些功能的应用程序进行测试，因为 ADL 并未复制整个执行环境。

## Trace 语句

在桌面上运行您的移动应用程序时，会将 trace 输出打印到调试器或用于启动 ADL 的终端窗口上。在设备或仿真器上运行应用程序时，可以设置远程调试会话以查看 trace 输出。如果支持由设备或操作系统制造商提供的软件开发工具，也可以使用这些工具来查看 trace 输出。

在任何情况下，都必须在启用调试功能的状态下编译应用程序中的 SWF 文件，以便运行时可输出任何 trace 语句。

### **Android** 上的远程 **trace** 语句

在 Android 设备或仿真器上运行时，可以使用 Android SDK 中包括的 Android Debug Bridge (ADB) 实用程序在 Android 系统日志中查看 trace 语句输出。若要查看应用程序的输出，请从开发计算机上的命令提示符下或终端窗口中运行下列命令：

```
tools/adb logcat air.MyApp:I *:S
```

其中，MyApp 是您应用程序的 AIR 应用程序 ID。参数 *:S 禁止从所有其他程序输出。除 trace 输出之外，若要查看有关您的应用程序的系统信息，可以将 ActivityManager 包括在 Logcat 滤镜规范中：

```
tools/adb logcat air.MyApp:I ActivityManager:I *:S
```

这些命令示例假设您正在从 Android SDK 文件夹运行 ADB 或者已将 SDK 文件夹添加到路径环境变量中。

注：在 AIR 2.6+ 中， ADB 实用程序包含在 AIR SDK 中，位于 lib/android/bin 文件夹。

### **iOS** 上的远程 **trace** 语句

若要通过在 iOS 设备上运行的应用程序来查看 trace 语句的输出，您必须使用 Flash 调试器 (FDB) 建立远程调试会话。

#### 更多帮助主题

Android Debug Bridge：启用 Logcat 日志记录

第 237 页的 " 路径环境变量 "

## 连接到 Flash 调试器

若要在移动设备上调试运行的应用程序，可以在您的开发计算机上运行 Flash 调试器并通过网络与其进行连接。若要启用远程调试，必须执行以下操作：

- 在 Android 上，在应用程序描述符中指定 android:permission.INTERNET 权限。
- 在启用调试功能的状态下编译应用程序 SWF。
- 打包应用程序 （对于 Android，请使用 -target apk-debug ；对于 iOS，请使用 -target ipa-debug）和 -connect 标志。

远程调试通过网络连接 （而不是 USB）进行，因此设备必须能够通过 IP 地址或标准域名访问运行 Flash 调试器的计算机的 TCP 端口 7935。

## 使用 Flash Professional 进行远程调试

在您的应用程序准备好调试并且在应用程序描述符中设置权限之后，请执行以下操作：

1 打开 "AIR Android 设置 " 对话框。

2 在 " 部署 " 选项卡下：

- 针对部署类型选择 " 设备调试 "
- 针对 " 发布后 " 选择 " 在连接的 Android 设备上安装应用程序 "
- 针对 " 发布后 " 取消选择 " 在连接的 Android 设备上启动应用程序 "
- 如果需要，将路径设为 Android SDK。

3 单击 " 发布 "。

   您的应用程序即会在设备上安装并启动。

4 关闭 "AIR Android 设置 " 对话框。

5 从 Flash Professional 菜单中选择 " 调试 ">" 开始远程调试会话 ">"ActionScript 3"。

   Flash Professional 会在 " 输出 " 面板中显示 " 正在等待播放器连接 "。

6 在设备上启动应用程序。

7 在 Adobe AIR 连接对话框中输入运行 Flash 调试器的计算机的 IP 地址或主机名，然后单击 " 确定 "。

## 通过网络连接使用 FDB 进行远程调试

若要使用命令行 Flash Debugger (FDB) 调试在设备上运行的应用程序，请首先在您的开发计算机上运行调试器，然后在设备上启动应用程序。下列过程使用 AMXMLC、 FDB 和 ADT 工具在设备上进行应用程序的编译、打包和调试。这些示例假设您组合使用 Flex 和 AIR SDK，并且 bin 目录包括在您的路径环境变量中。（此项假设仅仅是为了简化命令示例。）

1 打开终端或命令提示窗口并导航到包含应用程序源代码的目录。

2 使用 amxmlc 编译应用程序，从而能够进行调试：

```
amxmlc -debug DebugExample.as
```

3 使用 apk-debug 或 ipa-debug 目标打包应用程序：

```
Android
adt -package -target apk-debug -connect -storetype pkcs12 -keystore ../../AndroidCert.p12
DebugExample.apk DebugExample-app.xml DebugExample.swf

iOS
adt -package -target ipa-debug -connect -storetype pkcs12 -keystore ../../AppleDeveloperCert.p12 -
provisioning-profile test.mobileprovision DebugExample.apk DebugExample-app.xml DebugExample.swf
```

   如果您始终使用相同的主机名或 IP 地址进行调试，则可以将该值放在 -connect 标志之后。应用程序将尝试自动连接到该 IP 地址或主机名。否则，每次您开始调试时都必须在设备上输入信息。

4 安装应用程序。

   在 Android 上，您可以使用 ADT -installApp 命令：

```
adt -installApp -platform android -package DebugExample.apk
```

   在 iOS 上，您可以使用 iTunes 安装应用程序。

5 在另一个终端或命令窗口，运行 FDB：

```
fdb
```

6 在 FDB 窗口中，键入 run 命令：

```
Adobe fdb (Flash Player Debugger) [build 14159]
Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
(fdb) run
Waiting for Player to connect
```

7 在设备上启动应用程序。

8 应用程序在设备或模拟器上启动之后，即会打开 Adobe AIR 连接对话框。（如果您在对应用程序进行打包时已使用 -connect 选项指定主机名或 IP 地址，则它将尝试使用该地址自动连接。）输入适当的地址并点击 " 确定 "。

   若要在此模式下连接到调试器，设备必须能够解析地址或主机名并连接到 TCP 端口 7935。需要网络连接。

9 当远程运行时连接到调试器时，您可以使用 FDB break 命令设置断点，然后使用 continue 命令开始执行：

```
(fdb) run
Waiting for Player to connect
Player connected; session starting.
Set breakpoints and then type 'continue' to resume the session.
[SWF] Users:juser:Documents:FlashProjects:DebugExample:DebugExample.swf - 32,235 bytes after
decompression
(fdb) break clickHandler
Breakpoint 1 at 0x5993: file DebugExample.as, line 14
(fdb) continue
```

## 通过 USB 使用 FDB 进行远程调试

**AIR 2.6**，仅限 **Android**

若要通过 USB 连接调试应用程序，您必须使用 -listen 选项而不是 -connect 选项打包应用程序。指定 -listen 选项后，当您启动应用程序时，运行时会侦听 TCP 端口 7936 上 Flash 调试器 (FDB) 的连接。利用 -p 选项运行 FDB，令 FDB 启动连接。为了让在桌面计算机上运行的 Flash 调试器连接到在设备或模拟器上运行的 AIR 运行时，您必须使用从 Android SDK 中获取的 Android Debug Bridge (ADB) 将设备端口转发到桌面端口。

1 打开终端或命令提示窗口并导航到包含应用程序源代码的目录。

2 使用 amxmlc 编译应用程序，从而能够进行调试：

```
amxmlc -debug DebugExample.as
```

3 使用 apk-debug 目标打包应用程序，并指定 -listen 选项：

```
adt -package -target apk-debug -listen -storetype pkcs12 -keystore ../../AndroidCert.p12 DebugExample.apk
DebugExample-app.xml DebugExample.swf
```

4 使用 USB 电缆将设备连接到调试计算机。（您还可以使用此过程调试在模拟器中运行的应用程序，在这种情况下， USB 连接就是不必要的 — 或不可能的。）

5 安装应用程序。

   您可以使用 ADT -installApp 命令：

```
adt -installApp -platform android -package DebugExample.apk
```

6 使用 Android ADB 实用程序将 TCP 端口 7936 从设备或模拟器转发到计算机：

```
adb forward tcp:7936 tcp:7936
```

7 在设备上启动应用程序。

8 在一个终端或命令窗口中使用 -p 选项运行 FDB：

```
fdb -p 7936
```

9 在 FDB 窗口中，键入 run 命令：

```
Adobe fdb (Flash Player Debugger) [build 14159]
Copyright (c) 2004-2007 Adobe, Inc. All rights reserved.
(fdb) run
```

**10** FDB 实用程序尝试连接到该应用程序。

**11** 当建立远程连接时，您可以使用 FDB break 命令设置断点，然后使用 continue 命令开始执行：

```
(fdb) run
Player connected; session starting.
Set breakpoints and then type 'continue' to resume the session.
[SWF] Users:juser:Documents:FlashProjects:DebugExample:DebugExample.swf - 32,235 bytes after
decompression
(fdb) break clickHandler
Breakpoint 1 at 0x5993: file DebugExample.as, line 14
(fdb) continue
```

注：端口号 7936 用作默认端口，可以供 AIR 运行时和 FDB 进行 USB 调试。您可以使用 ADT -listen 端口参数和 FDB -p 端口参数指定不同的端口。在这种情况下，您必须使用 Android Debug Bridge 实用程序将 ADT 中指定的端口号转发到 FDB 中指定的端口：adb forward tcp:adt_listen_port# tcp:fdb_port#

# 在移动设备上安装 AIR 和 AIR 应用程序

应用程序最终用户可以使用各自设备的标准应用程序和分发机制安装 AIR 运行时和 AIR 应用程序。

例如，在 Android 上，用户可以从 Android Market 安装应用程序。或者，如果已在"应用程序"设置中允许从未知源安装应用程序，则用户可以通过单击 Web 页上的链接或通过将应用程序包复制到自己的设备并将其打开来安装应用程序。如果用户尝试安装 Android 应用程序，但尚未安装 AIR 运行时，则系统会自动将用户导向可以安装运行时的 Market。

在 iOS 上，有两种方法可以向最终用户分发应用程序。主要分发渠道为 Apple 应用程序库。您还可以使用临时分发，以便允许有限数量的用户无需通过应用程序库即可安装您的应用程序。

## 安装用于开发的 AIR 运行时和应用程序

由于移动设备上的 AIR 应用程序安装为本机包，您可以使用常规平台工具来安装要测试的应用程序。在支持的情况下，您可以使用 ADT 命令安装 AIR 运行时和 AIR 应用程序。目前， Android 支持此方法。

在 iOS 上，您可以使用 iTunes 安装要测试的应用程序。测试应用程序必须使用 Apple 为应用程序开发特别颁发的代码签名证书进行签名，并使用开发供给配置文件打包。 AIR 应用程序是 iOS 上的自包含包。未使用单独的运行时。

### 使用 ADT 安装 AIR 应用程序

开发 AIR 应用程序时，可以使用 ADT 安装和卸载运行时和应用程序。(您的 IDE 也可以集成这些命令，这样就不必自己运行 ADT。)

可以使用 AIR ADT 实用程序在设备或仿真器上安装 AIR 运行时。必须安装提供给设备的 SDK。使用 -installRuntime 命令：

```
adt -installRuntime -platform android -device deviceID -package path-to-runtime
```

如果未指定 -package 参数，则会从已安装的 AIR SDK 中提供的运行时包中选择适用于设备或模拟器的包。

若要安装 AIR 应用程序，请使用类似的 -installApp 命令：

```
adt -installApp -platform android -device deviceID -package path-to-app
```

如果只连接和运行了一台设备或模拟器，则可以省略 -device 标志。为 -platform 参数设置的值应与在其上进行安装的设备相匹配。目前，唯一支持的值是 android。

注：重新安装之前必须删除现有的 AIR 运行时版本或 AIR 应用程序。

在 **iOS** 设备上安装 **AIR** 应用程序

若要在 iOS 设备上安装 AIR 应用程序进行测试：

1 打开 iTunes 应用程序。

2 如果您尚未执行此操作，请将该应用程序的供给配置文件添加到 iTunes。在 iTunes 中，选择 " 文件 ">" 添加到资料库 "。然后选择供给配置文件 （其文件类型为 mobileprovision）。

3 如果已安装相同版本的应用程序， iTunes 的某些版本不会替换该应用程序。在这种情况下，从您的设备和 iTunes 中的应用程序列表中删除该应用程序。

4 双击您的应用程序的 IPA 文件。此时，您的应用程序应显示在 iTunes 中的应用程序列表中。

5 将您的设备连接到计算机上的 USB 端口。

6 在 iTunes 中，检查 " 应用程序 " 选项卡中是否存在该设备，并确保在要安装的应用程序的列表中选中了该应用程序。

7 选择左侧 iTunes 应用程序列表中的设备。然后单击 " 同步 " 按钮。完成同步后， Hello World 应用程序会显示在您的 iPhone 上。

如果未安装新版本，请将其从您的设备以及 iTunes 中的应用程序列表中删除，然后重新执行此过程。这可能是因为目前安装的版本使用的是相同的应用程序 ID 和版本。

**更多帮助主题**

第 133 页的 "ADT installRuntime 命令 "

第 131 页的 "ADT installApp 命令 "

## 在设备上运行 AIR 应用程序

可以使用设备用户界面启动已安装的 AIR 应用程序。在支持的情况下，也可以使用 AIR ADT 实用程序远程启动应用程序：

```
adt -launchApp -platform android -device deviceID -appid applicationID
```

-appid 参数值必须是要启动的 AIR 应用程序 ID。使用 AIR 应用程序描述符中指定的值 （不带打包期间添加的 air. 前缀）。

如果只连接和运行了一台设备或模拟器，则可以省略 -device 标志。为 -platform 参数设置的值应与在其上进行安装的设备相匹配。目前，唯一支持的值是 android。

## 删除 AIR 运行时和应用程序

可以使用设备操作系统提供的常规方式删除应用程序。在支持的情况下，您还可以使用 AIR ADT 实用程序删除 AIR 运行时和应用程序。若要删除运行时，请使用 -uninstallRuntime 命令：

```
adt -uninstallRuntime -platform android -device deviceID
```

若要卸载某个应用程序，请使用 -uninstallApp 命令：

```
adt -uninstallApp -platform android -device deviceID -appid applicationID
```

如果只连接和运行了一台设备或模拟器，则可以省略 -device 标志。为 -platform 参数设置的值应与在其上进行安装的设备相匹配。目前，唯一支持的值是 android。

## 设置仿真器

若要在设备仿真器上运行 AIR 应用程序，通常必须使用该设备的 SDK 在开发计算机上创建和运行仿真器实例。然后可以在仿真器上安装 AIR 运行时的仿真器版本和 AIR 应用程序。请注意，应用程序在仿真器上的运行速度通常比在实际设备上慢得多。

### 创建 Android 仿真器

1 启动 Android SDK 和 AVD Manager 应用程序：
- 在 Windows 上，请在 Android SDK 根目录中运行 SDK Setup.exe 文件。
- 在 Mac OS 上，请在 Android SDK 目录的工具子目录中运行 android 应用程序。

2 选择 Settings 选项和 “Force https://” 选项。

3 选择 Available Packages 选项。可以看见可用的 Android SDK 列表。

4 选择兼容的 Android SDK （Android 2.2 或更新版本），单击 Install Selected 按钮。

5 选择 Virtual Devices 选项，然后单击 New 按钮。

6 进行以下设置：
- 虚拟设备的名称
- 目标 API，例如 Android 2.2， API 级别 8
- SD 卡的大小 （如 1024）
- 外观 （例如，默认 HVGA）

7 单击 Create AVD 按钮。

请注意，创建 Virtual Device 可能需要一些时间，具体情况取决于系统配置。

现在可以启动新的 Virtual Device 了。

1 在 AVD Manager 应用程序中选择 Virtual Device。系统应列出前面创建的虚拟设备。

2 选择 Virtual Device，然后单击 Start 按钮。

3 单击下一屏幕上的 Launch 按钮。

此时会看见桌面上打开一个仿真器窗口。此过程需要几秒钟。 Android 操作系统初始化也可能需要一些时间。您可以在仿真器上安装使用 apk-debug 和 apk-emulator 进行打包的应用程序。使用 apk 目标打包的应用程序在仿真器上不起作用。

**更多帮助主题**

http://developer.android.com/guide/developing/tools/othertools.html#android

http://developer.android.com/guide/developing/tools/emulator.html

## 更新移动 AIR 应用程序

移动 AIR 应用程序作为本机软件包进行分发，因此将使用平台上其他应用程序的标准更新机制。通常，这涉及提交到相同的 Market 或用于分发原始应用程序的应用程序库。

移动 AIR 应用程序不能使用 AIR Updater 类或框架。

### 更新 Android 上的 AIR 应用程序

对于在 Android Market 上发布的应用程序，只要下列各项全部满足 （这些策略由 Market 而不是 AIR 强制执行），您就可以通过在 Market 上放置新的版本来更新应用程序：

- 使用同一个证书对 APK 包签名。
- AIR ID 相同。
- 应用程序描述符中的 versionNumber 值更大。（如果使用 versionLabel 值，还应增大该值。）

设备软件会向从 Android Market 下载应用程序的用户通知可用更新。

**更多帮助主题**

Android 开发人员：在 Android Market 上发布更新

## 更新 iOS 上的 AIR 应用程序

对于通过 iTunes 应用程序库分发的 AIR 应用程序，只要满足下列条件 （这些策略是由 Apple 应用程序库强制实施的，而不是 AIR）即可通过将更新提交到库来更新应用程序：

- 代码签名证书和供给配置文件会颁发给同一 Apple ID
- IPA 包使用同一 Apple Bundle ID
- 更新不会减小支持设备池 （也就是说，如果原始应用程序支持运行 iOS 3 的设备，则不能创建不再支持 iOS 3 支持的更新）。

重要说明：由于 AIR SDK 版本 2.6 和更高版本不支持 iOS 3，但 AIR 2 支持，因此如果某个已发布 iOS 应用程序是使用 AIR 2 开发的，而其更新是使用 AIR 2.6 以上版本开发的，则无法更新该应用程序。

# 第 8 章：开发针对电视设备的 AIR 应用程序

您可以为包含用于电视的 Adobe AIR 的电视设备 （如电视机、数码录像机和蓝光播放机）创建 Adobe® AIR® 应用程序。针对电视设备的 AIR 应用程序是基于 SWF 的应用程序，而不是基于 HTML 的应用程序。用于电视的 AIR 针对电视设备进行了优化，利用设备的硬件加速等功能实现高性能的视频和图形显示。

开发 AIR 应用程序的这个过程在很大程度上与为任何其他设备开发 AIR 应用程序相同。主要不同点体现在以下方面：

- 目标设备组的功能差异。
- 特定于用于电视的 AIR 应用程序的设计注意事项。
- 如何在目标设备上进行测试。

## 设备功能

### 设备配置文件

AIR 使用配置文件来定义具有类似功能的一组目标设备。对于用于电视的 AIR 应用程序，请使用以下配置文件：

- tv 配置文件。在针对用于电视的 AIR 设备的 AIR 应用程序中使用此配置文件。
- extendedTV 配置文件。如果用于电视的 AIR 应用程序使用本机扩展，则使用此配置文件。

第 186 页的 " 设备配置文件 " 中介绍了针对这些配置文件定义的 ActionScript 功能。用于 Adobe Flash Platform 的 ActionScript 3.0 参考中介绍了用于电视的 AIR 的 ActionScript 的具体差异。

有关用于电视的 AIR 配置文件的详细信息，请参阅第 105 页的 " 支持的配置文件 "。

### 硬件加速

电视设备提供了可大幅提升 AIR 应用程序中图形和视频性能的硬件加速器。要使用这些硬件加速器，请参阅第 90 页的 " 用于电视的 AIR 应用程序设计注意事项 "。

### 内容保护

使用用于电视的 AIR 可以基于优质的视频内容 （从好莱坞大片到独立制片的电影和电视剧）创建丰富的用户体验。内容提供商可以使用 Adobe 工具创建交互式应用程序。他们可以将 Adobe 服务器产品集成到内容分发基础结构中，也可以与某个 Adobe 生态系统合作伙伴进行合作。

内容保护是优质视频分发的关键要求。用于电视的 AIR 支持 Adobe® Flash® Access™，这一内容保护和商业化解决方案可满足内容所有者 （包括各大电影制片厂）的严格安全要求。

Flash Access 支持以下功能：

- 视频流和下载。
- 各种商业模式，包括支持广告、订阅、租赁和电子零售。
- 不同的内容分发技术，包括 HTTP 动态流、使用 Flash® Media Server 通过 RTMP （实时媒体协议）传输的流媒体以及通过 HTTP 渐进式下载。

用于电视的 AIR 还内置了对 RTMPE （RTMP 的加密版本）以及对安全性要求较低的现有流媒体解决方案的支持。 Flash Media Server 支持 RTMPE 和相关的 SWF 验证技术。

**更多帮助主题**

Adobe Flash Access

Flash Access 2.0 资源

使用 Adobe Flash Access 保护流媒体概述 (PDF)

使用 Adobe Flash 媒体技术保护在线视频分发

## 多频道音频

从 AIR 3 开始，用于电视的 AIR 为从 HTTP 服务器渐进式下载的视频提供多频道音频支持。此支持包括以下这些编解码器：

- AC-3 (Dolby Digital)
- E-AC-3 （增强的 Dolby Digital）
- DTS Digital Surround
- DTS Express
- DTS-HD High Resolution Audio
- DTS-HD Master Audio

注：从 Adobe Flash Media Server 流式传输的视频中的多频道音频支持尚不可用。

## 游戏输入

从 AIR 3 开始，用于电视的 AIR 支持 ActionScript API，允许应用程序与连接的游戏输入设备通信，比如游戏杆、游戏板和游戏棒。尽管这些设备称为游戏输入设备，但不仅仅是游戏，任何用于电视的 AIR 应用程序都可以使用这些设备。

可以使用多种具有不同功能的游戏输入设备。因此，在 API 中对这些设备进行了归纳，以便应用程序能够正常使用不同 （且可能是未知）类型的游戏输入设备。

GameInput 类是游戏输入 ActionScript API 的入口点。有关详细信息，请参阅 GameInput。

## Stage 3D 加速图形渲染

从 AIR 3 开始，用于电视的 AIR 支持 Stage 3D 加速图形渲染。Stage3D ActionScript API 是一组支持高级 2D 和 3D 功能的底层 GPU 加速 API。这些底层 API 为开发人员提供了灵活性，使其可以利用 GPU 硬件加速获取显著的性能提高。您还可以使用支持 Stage3D ActionScript API 的游戏引擎。

有关详细信息，请参阅游戏引擎、 3D 和 Stage 3D。

## 本机扩展

如果您的应用程序针对的是 extendedTV 配置文件，则可以使用 ANE （AIR 本机扩展）包。

通常，设备制造商会提供 ANE 包以用于访问 AIR 不支持的设备功能。例如，利用本机扩展可以更换电视频道或暂停视频播放器上的播放。

在对使用 ANE 包的用于电视的 AIR 应用程序进行打包时，必须将该应用程序打包为 AIRN 文件，而不是 AIR 文件。

用于电视的 AIR 设备的本机扩展始终是设备捆绑的 本机扩展。“ 设备捆绑的 ” 表示扩展库安装在用于电视的 AIR 设备上。您应用程序包中包含的 ANE 包从不 包含扩展的本机库。有时它包含纯 ActionScript 版本的本机扩展。此纯 ActionScript 版本是扩展的存根或模拟器。设备制造商将实际扩展 （其中包含本机库）安装到设备上。

如果您正在开发本机扩展，请注意以下事项：

- 如果您正在为制造商的设备创建用于电视的 AIR 本机扩展，请始终咨询设备制造商。

- 在某些用于电视的 AIR 设备上，只有设备制造商可以创建本机扩展。
- 在所有用于电视的 AIR 设备上，都由设备制造商决定需要安装哪些本机扩展。
- 构建用于电视的 AIR 本机扩展所使用的开发工具会有所不同，由制造商决定。

有关在您的 AIR 应用程序中使用本机扩展的详细信息，请参阅第 112 页的“使用 Adobe AIR 的本机扩展”。

有关创建本机扩展的信息，请参阅针对 Adobe AIR 开发本机扩展。

# 用于电视的 AIR 应用程序设计注意事项

## 视频注意事项

视频编码准则
在向电视设备传输视频流时， Adobe 建议采用以下编码原则：

| | |
|---|---|
| 视频编解码器： | H.264，主要或高级配置文件，渐进式编码 |
| 分辨率： | 720i、 720p、 1080i 或 1080p |
| 帧速率： | 每秒 24 帧或每秒 30 帧 |
| 音频编解码器： | AAC-LC 或 AC-3，44.1 kHz，立体声，或这些多频道音频编解码器：E-AC-3、DTS、DTS Express、 DTS-HD High Resolution Audio 或 DTS-HD Master Audio |
| 组合比特率： | 最高 8M bps，具体取决于可用带宽 |
| 音频比特率： | 最高 192 Kbps |
| 像素高宽比： | 1 × 1 |

对于传送到用于电视的 AIR 设备的视频， Adobe 建议使用 H.264 编解码器。

注：用于电视的 AIR 还支持使用 Sorenson Spark 或 On2 VP6 编解码器编码的视频。但是，硬件无法对这些编解码器进行解码并呈现。而是由运行时使用软件对这些编解码器进行解码并呈现，因此，视频会以相当低的帧速率进行播放。因此，请尽可能使用 H.264。

**StageVideo** 类
用于电视的 AIR 支持对 H.264 编码的视频进行硬件解码并呈现。使用 StageVideo 类可以启用此功能。

请参阅 ActionScript 3.0 开发人员指南中的使用 StageVideo 类实现硬件加速呈现，了解以下详细信息：

- StageVideo 类和相关类的 API。
- StageVideo 类的使用限制。

为了更好地支持使用 Video 对象播放 H.264 编码的视频的现有 AIR 应用程序，用于电视的 AIR 将在内部 使用 StageVideo 对象。这样做意味着可以利用硬件解码和呈现实现视频播放。但是， Video 对象与 StageVideo 对象具有相同的限制。例如，如果应用程序试图旋转视频，但却无法旋转，这是因为正在使用硬件而非运行时呈现视频。

不过，在编写新应用程序时，请使用 StageVideo 对象来播放 H.264 编码的视频。

有关使用 StageVideo 类的示例，请参阅在电视上传送 Flash Platform 的视频和内容。

视频传输指南

在用于电视的 AIR 设备上，视频播放期间网络的可用带宽可能会发生变化。例如，当另一个用户开始使用同一 Internet 连接时，可用带宽可能会发生变化。

因此， Adobe 建议在您的视频传输系统中采用自适应比特率功能。例如，在服务器端， Flash Media Server 支持自适应比特率功能。在客户端，可以使用开源媒体框架 (OSMF)。

可以采用以下协议将视频内容通过网络传送到用于电视的 AIR 应用程序：

- HTTP 和 HTTPS 动态流 （F4F 格式）
- RTMP、 RTMPE、 RTMFP、 RTMPT 和 RTMPTE 流
- HTTP 和 HTTPS 渐进式下载

**更多帮助主题**

Adobe Flash Media Server 开发人员指南

Open Source Media Framework

## 音频注意事项

在用于电视的 AIR 应用程序中，用于播放声音的 ActionScript 与其他 AIR 应用程序中的相应 ActionScript 没有任何不同。有关详细信息，请参阅 ActionScript 3.0 开发人员指南 中的使用声音。

关于用于电视的 AIR 中的多频道音频支持，请考虑以下事项：

- 用于电视的 AIR 为从 HTTP 服务器渐进式下载的视频提供多频道音频支持。从 Adobe Flash Media Server 流式传输的视频中的多频道音频支持尚不可用。
- 尽管用于电视的 AIR 支持许多音频编解码器，但并非所有的用于电视的 AIR 设备 都支持整套编解码器。使用 flash.system.Capabilities 方法 hasMultiChannelAudio() 检查用于电视的 AIR 设备是否支持特殊多频道音频编解码器（如 AC-3）。

  例如，考虑从服务器渐进式下载视频文件的应用程序。服务器具有支持不同多频道音频编解码器的各种 H.264 视频文件。应用程序可以使用 hasMultiChannelAudio() 确定要从服务器请求的视频文件。或者，应用程序可以向服务器发送 Capabilities.serverString 中包含的字符串。字符串指示可用的多频道音频编解码器，从而允许服务器选择适当的视频文件。
- 当使用某个 DTS 音频编解码器时，存在 hasMultiChannelAudio() 返回 true，但不播放 DTS 音频的情况。

  例如，假设带有 S/PDIF 输出的蓝光播放器连接到旧功放。旧功放不支持 DTS，但 S/PDIF 没有协议通知蓝光播放器。如果蓝光播放器向旧功放发送 DTS 流，则用户什么也听不到。因此，最佳做法是，在使用 DTS 时提供一个用户界面，以便该用户可以知道是否在播放声音。然后，应用程序可以还原到其他的编解码器。

下表概述了何时需要在用于电视的 AIR 应用程序中使用不同的音频编解码器。该表格还指示用于电视的 AIR 设备何时使用硬件加速器解码音频编解码器。硬件解码会提高性能并卸载 CPU。

| 音频编解码器 | 用于电视的 AIR 设备的可用性 | 硬件解码 | 何时使用此音频编解码器 | 更多信息 |
| --- | --- | --- | --- | --- |
| AAC | 始终为 | 始终为 | 在采用 H.264 编码的视频中。<br>对于 Internet 音乐流媒体服务等音频流。 | 当使用纯音频 AAC 流时，请将音频流封装在 MP4 容器中。 |
| mp3 | 始终为 | 否 | 对于应用程序的 SWF 文件中的声音。<br>在用 Sorenson Spark 或 On2 VP6 编码的视频中使用。 | 使用 mp3 音频的 H.264 视频在用于电视的 AIR 设备上无法播放。 |
| AC-3 (Dolby Digital)<br>E-AC-3 （增强的 Dolby Digital）<br>DTS Digital Surround<br>DTS Express<br>DTS-HD High Resolution Audio<br>DTS-HD Master Audio | 检查 | 是 | 在采用 H.264 编码的视频中。 | 通常，用于电视的 AIR 将多频道音频流传递到解码和播放音频的外部音频 / 视频接收器。 |
| Speex | 始终为 | 否 | 接收实时语音流。 | 使用 Speex 音频的 H.264 视频在用于电视的 AIR 设备上无法播放。 Speex 只能与 Sorenson Spark 或 On2 VP6 编码的视频一起使用。 |
| NellyMoser | 始终为 | 否 | 接收实时语音流。 | 使用 NellyMoser 音频的 H.264 视频在用于电视的 AIR 设备上无法播放。 NellyMoser 只能与 Sorenson Spark 或 On2 VP6 编码的视频一起使用。 |

注：某些视频文件包含两个音频流。例如，视频文件可以同时包含 AAC 流和 AC3 流。用于电视的 AIR 不支持此类视频文件，使用此类文件会导致视频没有声音。

## 图形硬件加速

使用硬件图形加速
用于电视的 AIR 设备为 2D 图形操作提供了硬件加速功能。设备的硬件图形加速器可以代替 CPU 执行以下操作：

- 位图呈现
- 位图缩放
- 位图混合
- 实心矩形填充

这种硬件图形加速意味着在用于电视的 AIR 应用程序中，可以更加高效地执行许多图形操作。其中的一些操作包括：

- 滑动转换
- 缩放转换
- 淡入淡出
- 使用 alpha 合成多个图像

若要获得此类操作的硬件图形加速性能优势，请使用以下方法之一：

- 将 MovieClip 对象和内容大部分均为发生更改的其他显示对象的 cacheAsBitmap 属性设置为 true。然后对这些对象进行滑动转换、淡入淡出转换和 alpha 混合。
- 在要缩放或平移 （应用 X 和 Y 重新定位）的显示对象上使用 cacheAsBitmapMatrix 属性。

  通过使用 Matrix 类操作进行缩放和平移，设备的硬件加速器会执行这些操作。此外，还可以考虑采用以下方案：更改 cacheAsBitmap 属性设置为 true 的显示对象的尺寸。当尺寸发生更改时，运行时软件会重新绘制位图。与使用 Matrix 操作的硬件加速进行缩放相比，使用软件进行重绘的性能要差一些。

  例如，假设有这样一个应用程序，当最终用户选择其中显示的图像时，该图像将扩大。多次使用 Matrix 缩放操作可营造图像逐渐扩大的效果。但是，如果原始图像和最终图像的大小相差过大，最终图像的品质可能无法令人满意。因此，请在扩大操作完成后重置显示对象的尺寸。由于 cacheAsBitmap 为 true，运行时软件会重新绘制显示对象 （但只执行一次），从而可呈现高品质的图像。

  注：用于电视的 AIR 设备通常不支持硬件加速的旋转和倾斜。因此，如果在 Matrix 类中指定旋转和倾斜，用于电视的 AIR 将使用软件执行所有 Matrix 操作。这些软件操作会对性能产生不利影响。
- 使用 BitmapData 类创建自定义位图缓存行为。

管理图形内存

要执行加速图形操作，硬件加速器会使用专用的图形内存。如果应用程序使用了所有图形内存，则应用程序的运行速度会变慢，因为用于电视的 AIR 会恢复为使用软件执行图形操作。

管理应用程序对图形内存的使用：

- 使用完图像或其他位图数据后，请释放相关的图形内存。若要释放图形内存，请调用 Bitmap 对象 bitmapData 属性的 dispose() 方法。例如：

  ```
  myBitmap.bitmapData.dispose();
  ```

  注：释放对 BitmapData 对象的引用不会立即释放图形内存。运行时的垃圾回收器最终会释放图形内存，但是调用 dispose() 可为应用程序提供更多控制手段。
- 使用 PerfMaster Deluxe （Adobe 提供的 AIR 应用程序）可更好地了解目标设备上的硬件图形加速功能。此应用程序可显示执行各种操作的每秒帧数。使用 PerfMaster Deluxe 可比较相同操作的不同实现。例如，比较移动位图图像与移动矢量图像。用于电视的 Flash Platform 提供了 PerfMaster Deluxe。

管理显示列表

要使某个显示对象不可见，请将该对象的 visible 属性设置为 false。则该对象仍在显示列表中，但用于电视的 AIR 不会呈现或显示它。此技术对于在视图中来回频繁显示的对象非常有用，因为它只产生少量的处理开销。但是，将 visible 属性设置为 false 不会释放任何对象资源。因此，当您完成某个对象的显示或至少将其显示较长一段时间时，请从显示列表中删除该对象。此外，将对该对象的所有引用设置为 null。这些操作允许垃圾回收器释放对象资源。

**更多帮助主题**

缓存显示对象

位图缓存

手动位图缓存

## PNG 和 JPEG 图像的使用

PNG 和 JPEG 是应用程序中常用的两种图像格式。关于用于电视的 AIR 应用程序中的这些图像格式，请考虑以下事项：

- 用于电视的 AIR 通常使用硬件加速来解码 JPEG 文件。

- 用于电视的 AIR 通常使用软件来解码 PNG 文件。使用软件解码 PNG 文件的速度非常快。
- PNG 是唯一支持透明度 （Alpha 通道）的跨平台位图格式。

因此，在应用程序中使用这些图像格式时，请遵循以下原则：

- 对于照片，请使用 JPEG 文件以享受硬件加速解码的优点。
- 对于用户界面元素，请使用 PNG 图像文件。可以对用户界面元素进行 alpha 设置，而且，对于用户界面元素，软件解码的性能已足够快。

## 用于电视的 AIR 应用程序中的舞台

若要创作用于电视的 AIR 应用程序，在使用 Stage 类时，请考虑以下事项：

- 屏幕分辨率
- 安全观看区域
- 舞台缩放模式
- 舞台对齐方式
- 舞台显示状态
- 多种屏幕尺寸设计
- 舞台品质设置

### 屏幕分辨率

目前，电视设备通常具有以下屏幕分辨率之一：540p、720p 和 1080p。这些屏幕分辨率对应于 ActionScript Capabilities 类中的以下值：

| 屏幕分辨率 | Capabilities.screenResolutionX | Capabilities.screenResolutionY |
|---|---|---|
| 540p | 960 | 540 |
| 720p | 1280 | 720 |
| 1080p | 1920 | 1080 |

若要针对特定设备编写用于电视的 AIR 全屏应用程序，请将 Stage.stageWidth 和 Stage.stageHeight 硬编码为设备的屏幕分辨率。但是，若要编写可在多种设备上运行的全屏应用程序，请使用 Capabilities.screenResolutionX 和 Capabilities.screenResolutionY 属性设置舞台尺寸。

例如：

```
stage.stageWidth = Capabilities.screenResolutionX;
stage.stageHeight = Capabilities.screenResolutionY;
```

### 安全观看区域

电视机上的安全观看区域 是指沿屏幕边缘向内缩放而形成的一个屏幕区域。该区域的内缩距离足够大，可确保最终用户能够看到整个区域，而不会被电视外壳的边缘所遮盖。由于电视外壳边缘 （形成屏幕的物理边框）的宽度因制造商而异，因此所需的内缩距离也会有所不同。安全观看区域试图保证屏幕可见区域的准确性。安全观看区域也称为标题安全区域。

过扫描 区域是指被外壳边缘遮盖的不可见屏幕区域。

Adobe 建议在屏幕的每个边缘向内缩放 7.5%。例如：




1920 x 1080 屏幕分辨率的安全观看区域

在设计用于电视的 AIR 全屏应用程序时，应始终考虑安全观看区域：

- 将整个屏幕用作背景，如背景图像或背景颜色。
- 仅对关键应用程序元素 （如文本、图形、视频和诸如按钮之类的用户界面项目）使用安全观看区域。

下表显示了各种典型屏幕分辨率的安全观看区域尺寸 （使用 7.5% 内缩比例）。

| 屏幕分辨率 | 安全观看区域的宽度和高度 | 左边和右边内缩宽度 | 顶部和底部内缩高度 |
|---|---|---|---|
| 960 x 540 | 816 x 460 | 72 | 40 |
| 1280 x 720 | 1088 x 612 | 96 | 54 |
| 1920 x 1080 | 1632 x 918 | 144 | 81 |

但是，最佳做法是始终动态计算安全观看区域的尺寸。例如：

```
var horizontalInset, verticalInset, safeAreaWidth, safeAreaHeight:int;

horizontalInset = .075 * Capabilities.screenResolutionX;
verticalInset = .075 * Capabilities.screenResolutionY;
safeAreaWidth = Capabilities.screenResolutionX - (2 * horizontalInset);
safeAreaHeight = Capabilities.screenResolutionY - (2 * verticalInset);
```

### 舞台缩放模式

将 Stage.scaleMode 设置为 StageScaleMode.NO_SCALE，并侦听舞台的 resize 事件。

```
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.addEventListener(Event.RESIZE, layoutHandler);
```

此设置使舞台坐标与像素坐标相同。将此设置与 FULL_SCREEN_INTERACTIVE 显示状态和 TOP_LEFT 舞台对齐方式结合使用，可以有效地使用安全观看区域。

具体来说，在全屏应用程序中，此缩放模式意味着 Stage 类的 stageWidth 和 stageHeight 属性与 Capabilities 类的 screenResolutionX 和 screenResolutionY 属性相对应。

此外，当应用程序的窗口大小发生更改时，舞台内容会保持所定义的大小。运行时不执行任何自动布局或缩放。此外，当窗口大小发生更改时，运行时会调度 Stage 类的 resize 事件。因此，当启动应用程序时以及当调整应用程序窗口大小时，您可以完全控制如何调整应用程序的内容。

注：NO_SCALE 行为与任何 AIR 应用程序的行为相同。但是，在用于电视的 AIR 应用程序中，使用此设置对使用安全观看区域来说至关重要。

### 舞台对齐方式

将 Stage.align 设置为 StageAlign.TOP_LEFT：

```
stage.align = StageAlign.TOP_LEFT;
```

此对齐方式将 0,0 坐标放置在屏幕的左上角，这便于使用 ActionScript 放置内容。

将此设置与 NO_SCALE 缩放模式和 FULL_SCREEN_INTERACTIVE 显示状态结合使用，可以有效地使用安全观看区域。

### 舞台显示状态

在用于电视的 AIR 全屏应用程序中，将 Stage.displayState 设置为 StageDisplayState.FULL_SCREEN_INTERACTIVE：

```
stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
```

设置该值后， AIR 应用程序将扩展舞台以覆盖整个屏幕 （允许用户输入）。

Adobe 建议您使用 FULL_SCREEN_INTERACTIVE 设置。将此设置与 NO_SCALE 缩放模式和 TOP_LEFT 舞台对齐方式结合使用，可以有效地使用安全观看区域。

因此，对于全屏应用程序，在主文档类的 ADDED_TO_STAGE 事件的处理函数中，请执行以下操作：

```
private function onStage(evt:Event):void
{
    stage.scaleMode = StageScaleMode.NO_SCALE;
    stage.align = StageAlign.TOP_LEFT;
    stage.addEventListener(Event.RESIZE, onResize);
    stage.displayState = StageDisplayState.FULL_SCREEN_INTERACTIVE;
}
```

然后，在 RESIZE 事件的处理函数中：

- 将屏幕分辨率的大小与舞台宽度和高度相比较。如果相同，则引发 RESIZE 事件，因为舞台 示状态改变为 FULL_SCREEN_INTERACTIVE。
- 计算并保存安全观看区域的尺寸以及对应的内缩比例。

```
private function onResize(evt:Event):void
{
    if ((Capabilities.screenResolutionX == stage.stageWidth) &&
        (Capabilities.screenResolutionY == stage.stageHeight))
    {

        // Calculate and save safe viewing area dimensions.
    }
}
```

当舞台尺寸等于 Capabilities.screenResolutionX 和 screenResolutionY 时，用于电视的 AIR 会使硬件为您的视频和图形传递可能的最佳保真度。

注：电视屏幕上图形和视频的显示保真度可能与 Capabilities.screenResolutionX 和 screenResolutionY 值不同，具体取决于运行用于电视的 AIR 的设备。例如，运行用于电视的 AIR 的机顶盒的屏幕分辨率为 1280 x 720，而相连电视的屏幕分辨率为 1920 x 1080。不过，用于电视的 AIR 会使硬件传递可能的最佳保真度。因此，在本示例中，硬件会使用 1920 x 1080 的屏幕分辨率显示 1080p 视频。

多种屏幕尺寸设计

您可以按相同方式开发可在多种用于电视的 AIR 设备上正常运行并显示的用于电视的 AIR 全屏应用程序。请执行以下操作：

1 将舞台属性 scaleMode、 align 和 displayState 分别设置为推荐值：StageScaleMode.NO_SCALE、 StageAlign.TOP_LEFT 和 StageDisplayState.FULL_SCREEN_INTERACTIVE。
2 根据 Capabilities.screenResolutionX 和 Capabilities.screenResolutionY 设置安全观看区域。
3 根据安全观看区域的宽度和高度调整您的内容的大小布局。

   虽然您的内容对象非常庞大 （尤其是与移动设备应用程序相比），但是诸如动态布局、相对定位和自适应内容之类的概念是相同的。有关 ActionScript 对这些概念的支持的详细信息，请参阅针对多种屏幕大小创作移动 Flash 内容。

舞台品质

用于电视的 AIR 应用程序的 Stage.quality 属性始终为 StageQuality.High。您无法更改它。

此属性指定所有舞台对象的呈现品质。

## 遥控器输入处理

用户通常使用遥控器与用于电视的 AIR 应用程序进行交互。但是，应像在桌面应用程序上处理键盘按键输入时一样，采用相同方式处理按键输入。具体来说，处理事件 KeyboardEvent.KEY_DOWN。有关更多信息，请参阅 ActionScript 3.0 开发人员指南 中的捕获键盘输入。

遥控器上的按键会映射到 ActionScript 常量。例如，遥控器键盘上的方向键映射如下：

| 遥控器键盘的方向键 | ActionScript 3.0 常量 |
| --- | --- |
| 向上键 | Keyboard.UP |
| 向下键 | Keyboard.DOWN |
| 向左键 | Keyboard.LEFT |
| 向右键 | Keyboard.RIGHT |
| “ 确定 ” 或 “ 选择 ” 键 | Keyboard.ENTER |

AIR 2.5 增加了许多其他 Keyboard 常量以支持遥控器输入。有关完整列表，请参阅用于 Adobe Flash Platform 的 ActionScript 3.0 参考 中的 Keyboard 类。

为确保您的应用程序可在尽可能多的设备上运行， Adobe 建议您参考以下做法：

- 如果可能，仅使用键盘方向键。

  不同的遥控设备具有不同的按键集。但是，它们通常始终包含键盘方向键。

  例如，蓝光播放器的遥控器通常不会有 “ 频道递增 ” 和 “ 频道递减 ” 键。甚至并非所有的遥控器上都有播放、暂停和停止键。

- 如果应用程序还需要键盘方向键以外的其他按键，请使用 “ 菜单 ” 和 “ 信息 ” 键。

  “ 菜单 ” 和 “ 信息 ” 键是遥控器上最常用的第二类按键。

- 考虑通用遥控器的频繁使用。

  即使您正在针对特定设备创建应用程序，也要认识到许多用户并不使用设备附带的遥控器，而是改用通用遥控器。此外，用户并不总是设置他们的通用遥控器来匹配设备遥控器上的所有按键。因此，最好只使用最常用的按键。

- 确保在任何情况下，用户始终可使用其中一个键盘方向键退出当前环境。

  有时，您的应用程序可能必须使用遥控器上的不常用按键。使用某个键盘方向键提供一种退出方案，可使您的应用程序在所有设备上都能正常使用。

- 请勿要求指针输入，除非您知道用于电视的 AIR 目标设备具有指针输入功能。

  虽然许多桌面应用程序期望鼠标输入，但是多数电视不支持指针输入。因此，如果您正在转换桌面应用程序以便其能够在电视机上运行，请确保您将应用程序修改为不要求鼠标输入。这些修改包括对事件处理的更改和对用户使用说明的更改。例如，当显示应用程序的启动屏幕时，不显示 " 点击开始 " 文本。

## 管理焦点

当桌面应用程序的某个用户界面元素获得焦点时，它将成为用户输入事件 （如键盘和鼠标事件）的目标。此外，应用程序会加亮显示获得焦点的用户界面元素。在用于电视的 AIR 应用程序中管理焦点与在桌面应用程序中管理焦点有所不同，原因如下：

- 桌面应用程序通常使用 tab 键将焦点更改到下一个用户界面元素上。使用 tab 键不适用于用于电视的 AIR 应用程序。遥控设备通常不具有 tab 键。因此，像在桌面上一样使用 DisplayObject 的 tabEnabled 属性来管理焦点并不适用。
- 桌面应用程序通常期望用户使用鼠标为用户界面元素提供焦点。

因此，请在您的应用程序中执行以下操作：

- 为舞台添加可侦听 Keyboard 事件 （例如， KeyboardEvent.KEY_DOWN）的事件侦听器。
- 提供应用程序逻辑以确定向最终用户加亮显示的用户界面元素。当启动应用程序后，确保加亮显示某个用户界面元素。
- 根据应用程序逻辑，将舞台接收到的 Keyboard 事件调度到相应的用户界面元素对象。

  也可以使用 Stage.focus 或 Stage.assignFocus() 将焦点分配给用户界面元素。然后，可以为该 DisplayObject 添加事件侦听器，以便该对象可接收键盘事件。

## 用户界面设计

通过采纳针对以下各项提供的建议，使用于电视的 AIR 应用程序的用户界面能够在电视机上正常工作：

- 应用程序的响应性能
- 应用程序的易用性
- 用户的个性和期望

### 响应性能

使用以下技巧尽量提高用于电视的 AIR 应用程序的响应能力。

- 使应用程序的初始 SWF 文件尽可能小。

  在初始 SWF 文件中，仅加载启动应用程序所必需的资源。例如，仅加载应用程序的启动屏幕图像。

  虽然此建议同样适用于桌面 AIR 应用程序，但在用于电视的 AIR 设备中更为重要。例如，用于电视的 AIR 设备不具备与桌面计算机同等的处理能力。此外，这些设备将应用程序存储在闪存中，访问闪存要比访问桌面计算机中的硬盘快得多。

- 以至少每秒 20 帧的帧速率运行应用程序。

  对图形进行专门设计以实现此目标。图形操作的复杂程度会影响每秒的帧数。有关改善呈现效果的技巧，请参阅优化 Adobe Flash Platform 的性能。

  注: 用于电视的 AIR 设备上的图形硬件通常以 60 Hz 或 120 Hz (每秒 60 或 120 次) 的速度更新屏幕。例如，对于 60 Hz 或 120 Hz 屏幕，硬件以每秒 30 帧或每秒 60 帧扫描舞台进行更新以显示画面。但是，用户能否体验这些较高的帧速率取决于应用程序图形操作的复杂程度。

- 在用户输入内容之后，在 100 至 200 毫秒内更新屏幕。

  如果需要更长时间才能完成更新，用户会变得不耐烦，往往导致多次按下按键。

易用性

用于电视的 AIR 应用程序的用户处于 " 客厅 " 环境中。他们坐在电视机的对面，离电视机大约 3 米远。客厅内有时会很黑暗。用户通常使用遥控设备进行输入。会有多个人使用应用程序，有时同时使用，有时交替使用。

因此，为改善用户界面在电视上的易用性，请在设计时考虑以下事项：

- 将用户界面元素放大。

  在设计文本、按钮或任何其他用户界面元素时，应假定用户坐在房间的对侧。使所有界面元素均能够在距离电视 3 米远的位置清楚观看和阅读。不要因屏幕够大而草率地让界面元素占据过多屏幕空间。

- 使用适当的对比度，以便能够坐在房间对侧轻松观看和阅读内容。
- 通过加亮显示具有焦点的用户界面元素使其突显。
- 仅在必要时使用动画。例如，连续从一个屏幕滑动到下一个屏幕可以正常工作。但是，如果动画不能帮助用户进行导航，或者动画不是应用程序所固有的，则会分散用户的注意力。
- 始终为用户提供通过用户界面返回的明显途经。

有关使用遥控器的更多信息，请参阅第 97 页的 " 遥控器输入处理 "。

用户的个性和期望

考虑到用于电视的 AIR 应用程序的用户通常是在娱乐性的轻松环境中寻求高品质的电视娱乐。他们不一定熟练掌握相关计算机或技术知识。

因此，在设计用于电视的 AIR 应用程序时，应遵循以下原则：

- 请勿使用技术术语。
- 避免使用模态对话框。
- 使用适用于客厅环境而非工作或技术环境的友好、通俗易懂的使用说明。
- 使用电视观众期望的具有较高产品品质的图形。
- 创建能够方便使用远程控制设备的用户界面。请勿使用更适用于桌面或移动应用程序的用户界面或设计元素。例如，桌面和移动设备上的用户界面通常涉及使用鼠标或手指指向并单击按钮。

## 字体和文本

可以在用于电视的 AIR 应用程序中使用设备字体，也可以使用嵌入字体。

设备字体是设备上安装的字体。所有用于电视的 AIR 设备都具有以下设备字体：

| 字体名称 | 说明 |
| --- | --- |
| _sans | _sans 设备字体是无衬线字体。所有用于电视的 AIR 设备上安装的 _sans 设备字体都是 Myriad Pro 字体。通常，由于观看距离的缘故，在电视上， sans-serif 字体比 serif 字体更美观。 |
| _serif | _serif 设备字体是有衬线字体。所有用于电视的 AIR 设备上安装的 _serif 设备字体都是 Minion Pro 字体。 |
| _typewriter | _typewriter 设备字体是等宽字体。所有用于电视的 AIR 设备上安装的 _typewriter 设备字体都是 Courier Std 字体。 |

所有用于电视的 AIR 设备还具有以下亚洲设备字体：

| 字体名称 | 语言 | 字体类别 | 区域设置代码 |
|---|---|---|---|
| RyoGothicPlusN-Regular | 日语 | sans | ja |
| RyoTextPlusN-Regular | 日语 | serif | ja |
| AdobeGothicStd-Light | 朝鲜语 | sans | ko |
| AdobeHeitiStd-Regular | 简体中文 | sans | zh_CN |
| AdobeSongStd-Light | 简体中文 | serif | zh_CN |
| AdobeMingStd-Light | 繁体中文 | serif | zh_TW 和 zh_HK |

这些用于电视的 AIR 设备字体均具有以下特征：

- 来自 Adobe® 类型库
- 在电视机上显示效果良好
- 针对视频字幕而设计
- 是轮廓字体，而不是位图字体

注：设备制造商通常会在设备上包含其他设备字体。除了用于电视的 AIR 设备字体之外，设备上还安装了这些制造商提供的设备字体。

Adobe 提供了一个名为 FontMaster Deluxe 的应用程序，可在设备上显示所有设备字体。可从用于电视的 Flash Platform 上获取该应用程序。

也可以在用于电视的 AIR 应用程序中嵌入字体。有关嵌入字体的信息，请参阅 ActionScript 3.0 开发人员指南 中的高级文本呈现。

在使用 TLF 文本字段时， Adobe 建议遵循以下原则：

- 对亚洲语言文本采用 TLF 文本字段，以利用运行应用程序所使用的区域设置。设置与 TLFTextField 对象相关联的 TextLayoutFormat 对象的 locale 属性。若要确定当前的区域设置，请参阅 ActionScript 3.0 开发人员指南 中的选择区域设置。
- 如果字体不是用于电视的 AIR 设备字体，请在 TextLayoutFormat 对象的 fontFamily 属性中指定字体名称。如果设备上提供了该字体，则用于电视的 AIR 将使用该字体。如果设备上未提供所请求的字体，则用于电视的 AIR 会根据 locale 设置使用相应的用于电视的 AIR 设备字体予以替换。
- 为 fontFamily 属性指定 _sans_serif 或 _typewriter 并设置 locale 属性，这样用于电视的 AIR 就能够选择正确的用于电视的 AIR 设备字体。根据区域设置，用于电视的 AIR 会从其亚洲设备字体集或非亚洲设备字体集中进行选择。这些设置为您提供一种简单的方法，可以自动为亚洲四种主要区域设置和英语使用正确的字体。

注：如果为亚洲语言文本使用传统文本字段，请指定用于电视的 AIR 设备字体的字体名称以保证正确呈现。如果知道您的目标设备上安装了另一种字体，也可以指定该字体。

关于应用程序性能，请考虑以下事项：

- 与 TLF 文本字段相比，传统文本字段能够提供更快的性能。
- 使用位图字体的传统文本字段能够提供最快的性能。

  与仅提供每个字符的轮廓数据的轮廓字体不同，位图字体可提供每个字符的位图。设备字体和嵌入字体均可为位图字体。
- 如果指定设备字体，请确保目标设备上已安装该设备字体。如果设备上未安装该设备字体，则用于电视的 AIR 会查找并使用设备上安装的其他字体。但是，此行为会降低应用程序的性能。
- 与所有显示对象一样，如果 TextField 对象基本未更改，请将该对象的 cacheAsBitmap 属性设置为 true。此设置可改善淡入淡出、滑动和 alpha 混合等转换效果。使用 cacheAsBitmapMatrix 进行缩放和转换。

## 文件系统安全

用于电视的 AIR 应用程序属于 AIR 应用程序，因此可以访问设备的文件系统。但是，对于 " 客厅 " 设备来说，应用程序不能访问该设备的文件系统或其他应用程序的文件，这一点至关重要。当电视及相关设备的用户正在观看电视时，他们不希望出现或者不能容忍任何设备故障，毕竟，他们是在观看电视。

因此，用于电视的 AIR 应用程序只能有限度地查看设备的文件系统。使用 ActionScript 3.0，应用程序只能访问特定目录（及其子目录）。此外，您在 ActionScript 中使用的目录名称并不是设备上的实际目录名称。这一附加保护层可防止用于电视的 AIR 应用程序恶意或意外地访问不属于它们的本地文件。

有关详细信息，请参阅使用用于电视的 AIR 应用程序查看目录。

## AIR 应用程序沙箱

用于电视的 AIR 应用程序在 AIR 应用程序沙箱中运行，如 AIR 应用程序沙箱中所述。

用于电视的 AIR 应用程序的唯一区别在于，它们只能有限制地访问文件系统 （如第 101 页的 " 文件系统安全 " 中所述）。

## 应用程序生命周期

与在桌面环境中不同的是，最终用户无法关闭正在运行用于电视的 AIR 应用程序的窗口。因此，必须提供退出应用程序的用户界面机制。

通常情况下，设备允许最终用户使用遥控器上的退出键无条件退出应用程序。但是，用于电视的 AIR 不会将 flash.events.Event.EXITING 事件调度到应用程序。因此，应经常保存应用程序状态，以便下次启动应用程序时可以自行恢复到合理状态。

## HTTP cookie

用于电视的 AIR 支持 HTTP 永久 ·cookie 和会话 cookie。用于电视的 AIR 将每个 AIR 应用程序的 cookie 存储在特定于应用程序的目录中：

```
/app-storage/<app id>/Local Store
```

Cookie 文件名为 cookies。

注：其他设备（如，桌面设备）上的 AIR 不会为每个应用程序单独存储 cookie。特定于应用程序的 cookie 存储支持用于电视的 AIR 的应用程序和系统安全模型。

请遵循以下原则使用 ActionScript 属性 URLRequest.manageCookies：

- 将 manageCookies 设置为 true。此值是默认值。它表示用于电视的 AIR 会自动向 HTTP 请求中添加 cookie，并记住 HTTP 响应中的 cookie。

  注：即使 manageCookies 设置为 true，应用程序也可以使用 URLRequest.requestHeaders 向 HTTP 请求中添加 cookie。如果此 cookie 与用于电视的 AIR 所管理的 cookie 具有相同名称，则请求中将包含同名的两个 cookie。两个 cookie 的值可以不同。

- 将 manageCookies 设置为 false。此值表示应用程序负责发送 HTTP 请求中的 cookie，并记住 HTTP 响应中的 cookie。

**更多帮助主题**

URLRequest

# 开发用于电视的 AIR 应用程序的工作流程

可以使用下列 Adobe Flash Platform 开发工具开发用于电视的 AIR 应用程序：

- Adobe Flash Professional

  Adobe Flash Professional CS5.5 支持用于电视的 AIR 2.5 （支持用于电视的 AIR 应用程序的第一个 AIR 版本）。

- Adobe Flash® Builder®

  Flash Builder 4.5 支持用于电视的 AIR 2.5。

- AIR SDK

  从 AIR 2.5 开始，可以使用 AIR SDK 提供的命令行工具开发应用程序。若要下载 AIR SDK，请参阅 http://www.adobe.com/products/air/sdk/。

## 使用 Flash Professional

使用 Flash Professional 开发、测试和发布用于电视的 AIR 应用程序与使用面向 AIR 桌面应用程序的工具类似。

但是，在编写 ActionScript 3.0 代码时，请仅使用 tv 和 extendedTV AIR 配置文件支持的类和方法。有关详细信息，请参阅第 186 页的 “ 设备配置文件 ”。

### 项目设置

执行以下操作以设置用于电视的 AIR 应用程序项目：

- 在 “ 发布设置 ” 对话框的 “Flash” 选项卡中，将 “ 播放器 ” 值至少设置为 AIR 2.5。
- 在 “Adobe AIR 设置 ” 对话框 （应用程序和安装程序设置）的 “ 常规 ” 选项卡中，将配置文件设置为 TV 或 extended TV。

### 调试

可以在 Flash Professional 中使用 AIR Debug Launcher 运行应用程序。请执行以下操作：

- 若要在调试模式下运行应用程序，请选择：

  “ 调试 ”>“ 调试影片 ”>“ 在 AIR Debug Launcher （桌面）中 ”

  做出此选择后，对于后续运行的调试可以选择：

  “ 调试 ”>“ 调试影片 ”>“ 调试 ”

- 若要不使用调试模式运行应用程序，请选择：

  “ 控制 ”>“ 测试影片 ”>“ 在 AIR Debug Launcher （桌面）中 ”

  做出此选择后，可以为后续运行选择 “ 控制 ”>“ 测试影片 ”>“ 测试 ”。

由于已将 AIR 配置文件设置为 TV 或 extendedTV， AIR Debug Launcher 将提供一个名为 “ 遥控器按钮 ” 的菜单。可以使用此菜单来模拟遥控设备上的按键。

### 使用本机扩展

如果应用程序使用本机扩展，请在库路径中包含本机扩展的 ANE 文件。然后可以使用 Flash Professional 来编译 ActionScript 代码。此过程的步骤位于第 114 页的 “ 在应用程序库路径中包含 ANE 文件 ”。

但是，当应用程序使用本机扩展时：

- 无法使用 Flash Professional 发布应用程序。若要发布应用程序，请使用 ADT。请参阅第 107 页的 “ 使用 ADT 打包 ”。

- 无法使用 Flash Professional 运行或调试应用程序。若要在开发计算机上调试应用程序，请使用 ADL。请参阅第 108 页的 “ 使用 ADL 的设备模拟 ”。

## 使用 Flash Builder

从 Flash Builder 4.5 开始，Flash Builder 支持用于电视的 AIR 开发。使用 Flash Builder 开发、测试和发布用于电视的 AIR 应用程序与使用面向 AIR 桌面应用程序的工具类似。

### 设置应用程序

请确保您的应用程序：

- 如果使用的是 MXML 文件，则使用 Application 元素作为 MXML 文件中的容器类：

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx">

    <!-- Place elements here. -->

</s:Application>.
```

  重要说明：用于电视的 AIR 应用程序不支持 WindowedApplication 元素。

  注：您完全不必使用 MXML 文件，可以创建 ActionScript 3.0 项目来代替。

- 仅使用 tv 和 extendedTV AIR 配置文件支持的 ActionScript 3.0 类和方法。有关详细信息，请参阅第 186 页的 “ 设备配置文件 ”。

此外，在应用程序的 XML 文件中，请确保：

- application 元素的 xmlns 属性设置为 AIR 2.5：

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

- supportedProfiles 元素包含 tv 或 extendedTV：

```
<supportedProfiles>tv</supportedProfiles>
```

### 调试应用程序

可以在 Flash Builder 中使用 AIR Debug Launcher 运行应用程序。请执行以下操作：

1 选择 “ 运行 ”>“ 调试配置 ”。

2 确保 “ 配置文件 ” 字段设置为 “ 桌面 ”。

3 选择 “ 运行 ”>“ 调试 ” 以在调试模式下运行，或者选择 “ 运行 ”>“ 运行 ” 不带调试模式功能运行。

由于已将 supportedProfiles 元素设置为 TV 或 extendedTV，AIR Debug Launcher 将提供一个名为 “ 遥控器按钮 ” 的菜单。可以使用此菜单来模拟遥控设备上的按键。

### 使用本机扩展

如果应用程序使用本机扩展，请在库路径中包含本机扩展的 ANE 文件。然后可以使用 Flash Builder 来编译 ActionScript 代码。此过程的步骤位于第 114 页的 “ 在应用程序库路径中包含 ANE 文件 ”。

但是，当应用程序使用本机扩展时：

- 无法使用 Flash Builder 发布应用程序。若要发布应用程序，请使用 ADT。请参阅第 107 页的 “ 使用 ADT 打包 ”。
- 无法使用 Flash Builder 运行或调试应用程序。若要在开发计算机上调试应用程序，请使用 ADL。请参阅第 108 页的 “ 使用 ADL 的设备模拟 ”。

# 针对 TV 应用程序属性设置 AIR

对于其他 AIR 应用程序，可以在应用程序描述符文件中设置基本应用程序属性。TV 配置文件应用程序会忽略某些特定于桌面的属性，例如窗口大小和透明度。面向 extendedTV 配置文件中的设备的应用程序可以使用本机扩展。这些应用程序可识别 extensions 元素中使用的本机扩展。

## 通用设置

某些应用程序描述符设置对所有 TV 配置文件应用程序都很重要。

### 所需的 AIR 运行时版本

使用应用程序描述符文件的命名空间指定应用程序所需的 AIR 运行时版本。

在 application 元素中分配的命名空间，很大程度上决定了应用程序可以使用哪些功能。例如，假设有一个应用程序使用的是 AIR 2.5 命名空间，但是用户却安装了某个未来版本。在这种情况下，应用程序仍遵循 AIR 2.5 行为，即使该行为在 AIR 的未来版本中已改变。只有当您更改命名空间并发布更新时，应用程序才会访问新的行为和功能。不过，安全修补程序不受此规则限制。

使用 application 根元素的 xmlns 属性指定命名空间：

```
<application xmlns="http://ns.adobe.com/air/application/2.5">
```

AIR 2.5 是第一个支持 TV 应用程序的 AIR 版本。

#### 更多帮助主题

第 158 页的 “application”

### 应用程序标识

对于发布的每个应用程序，以下几个设置应该是唯一的。这些设置包括 ID、名称和文件名。

```
<id>com.example.MyApp</id>
<name>My Application</name>
<filename>MyApplication</filename>
```

#### 更多帮助主题

第 169 页的 “id”

第 165 页的 “filename”

第 176 页的 “name”

### 应用程序版本

在 versionNumber 元素中指定应用程序版本。当为 versionNumber 指定值时，可以使用由点分隔的最多三个数字组成的序列，例如：“0.1.2”。版本号的每段最多可以具有三个数字。（即，“999.999.999” 是允许的最大版本号）。不必将所有三段都包含在号码中；“1” 和 “1.0” 都是合法的版本号。

也可以使用 versionLabel 元素来指定版本标签。如果添加版本标签，则会显示该版本标签而不是版本号。

```
<versionNumber>1.23.7<versionNumber>
<versionLabel>1.23 Beta 7</versionLabel>
```

### 更多帮助主题

第 182 页的 "version"

第 183 页的 "versionLabel"

第 183 页的 "versionNumber"

## 主应用程序 SWF

在 initalWindow 元素的 content 子元素中指定主应用程序 SWF 文件。在 TV 配置文件中定位设备时，必须使用 SWF 文件（不支持基于 HTML 的应用程序）。

```
<initialWindow>
    <content>MyApplication.swf</content>
</initialWindow>
```

必须包括 AIR 包中的文件 （使用 ADT 或 IDE）。如果只是引用应用程序描述符中的名称，不会自动将此文件包括进包中。

## 主屏幕属性

initialWindow 元素的一些子元素可控制主应用程序屏幕的初始外观和行为。尽管在 TV 配置文件中的设备上会忽略其中大多数属性，但是可以使用 fullScreen 元素：

- fullScreen — 指定应用程序是应占据设备的整个显示屏，还是与标准操作系统窗口样式共享显示屏。

  ```
  <fullScreen>true</fullScreen>
  ```

### 更多帮助主题

第 167 页的 "fullScreen"

## visible 元素

visible 元素是 initialWindow 元素的子元素。用于电视的 AIR 将忽略此 visible 元素，因为您应用程序的内容在用于电视的 AIR 设备上始终是可见的。

但是，如果您的应用程序也要面向桌面设备，则请将 visible 元素设置为 true。

在桌面设备上，此元素的值默认为 false。因此，如果不包含此 visible 元素，应用程序的内容在桌面设备上将不可见。虽然您可以通过 ActionScript 类 NativeWindow 使得应用程序内容在桌面设备上可见，但是电视设备的配置文件不支持 NativeWindow 类。如果您试图在某个应用程序上使用 NativeWindow 类，而此应用程序正运行在用于电视的 AIR 设备上，则此应用程序将加载失败。无论您是否调用 NativeWindow 类的方法都是如此；使用此类的应用程序在用于电视的 AIR 设备上无法加载。

## 支持的配置文件

如果应用程序仅用于电视设备，则可以防止在其他类型的计算设备上安装该应用程序。从支持的配置文件列表中排除其他配置文件：

```
<supportedProfiles>tv extendedTV</supportedProfiles>
```

如果应用程序使用本机扩展，请在支持的配置文件列表中只包含 extendedTV 配置文件：

```
<supportedProfiles>extendedTV</supportedProfiles>
```

如果忽略 supportedProfiles 元素，则会假定应用程序支持所有配置文件。

请勿在 supportedProfiles 列表中只 包含 tv 配置文件。一些电视设备始终以与 extendedTV 配置文件对应的模式运行用于电视的 AIR。正是由于这种行为才使得用于电视的 AIR 能够运行使用本机扩展的应用程序。如果您的 supportedProfiles 元素只指定了 tv，则表示您的内容与用于电视的 AIR 的 extendedTV 模式不兼容。因此，一些电视设备无法加载只指定了 tv 配置文件的应用程序。

有关 tv 和 extendedTV 配置文件中支持的 ActionScript 类列表，请参阅第 187 页的“ 不同配置文件的功能 ”。

## 必需的本机扩展

支持 extendedTV 配置文件的应用程序可以使用本机扩展。

在应用程序描述符中声明 AIR 应用程序使用的所有本机扩展。下面的例子说明了用于指定两个所需本机扩展的语法：

```
<extensions>
    <extensionID>com.example.extendedFeature</extensionID>
    <extensionID>com.example.anotherFeature</extensionID>
</extensions>
```

如果未列出某扩展，则应用程序将无法使用它。

extensionID 元素的值与扩展描述符文件中的 id 元素的值相同。扩展描述符文件是一个名为 extension.xml 的 XML 文件。已打包在从设备制造商处接收到的 ANE 文件中。

如果在 extensions 元素中列出了某扩展，但是用于电视的 AIR 设备未安装此扩展，则应用程序将无法运行。此规则的例外情况是，如果和用于电视的 AIR 应用程序打包在一起的 ANE 文件拥有扩展的存根版本，则此规则失效。如果是这种情况，应用程序将可以运行，它将使用扩展的存根版本。存根版本具有 ActionScript 代码，但无本机代码。

## 应用程序图标

对电视设备中应用程序图标的要求与设备有关。例如，设备制造商指定：

- 所需的图标和图标大小。
- 所需的文件类型和命名约定。
- 如何为应用程序提供图标，例如，是否将图标与应用程序一起打包。
- 是否在应用程序描述符文件的 <icon> 元素中指定图标。
- 当应用程序未提供图标时的行为。

有关详细信息，请咨询设备制造商。

### 更多帮助主题

第 168 页的“icon”

第 169 页的“imageNxN”

## 忽略的设置

电视设备上的应用程序会忽略应用于移动设备、本机窗口或桌面操作系统功能的应用程序设置。忽略的设置包括：

- allowBrowserInvocation
- aspectRatio
- autoOrients
- customUpdateUI
- fileTypes
- height
- installFolder
- maximizable

- maxSize
- minimizable
- minSize
- programMenuFolder
- renderMode
- resizable
- systemChrome
- title
- transparent
- visible
- width
- x
- y

# 对用于电视的 AIR 应用程序进行打包

## 使用 ADT 打包

可以使用 AIR ADT 命令行工具对用于电视的 AIR 应用程序进行打包。从 AIR SDK 版本 2.5 开始，ADT 支持为 TV 设备打包。在打包之前，首先编译所有 ActionScript 和 MXML 代码。还必须有代码签名证书。可以使用 ADT -certificate 命令创建证书。

有关 ADT 命令和选项的详细参考，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。

### 创建 AIR 包

若要创建 AIR 包，请使用 ADT package 命令：

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 myApp.air myApp-app.xml myApp.swf icons
```

此示例假设：

- 命令行 shell 的路径定义中已定义了 ADT 工具的路径。（请参阅第 237 页的 “ 路径环境变量 ”。）
- 证书 codesign.p12 位于从中运行 ADT 命令的父目录中。

从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、 myApp.swf 和图标目录。

当运行如上所示的命令时， ADT 会提示输入 keystore 密码。并非所有的 shell 程序都会显示键入的密码字符；只需在键入结束后按 Enter 即可。此外，也可以使用 storepass 参数将密码包含在 ADT 命令中。

### 创建 AIRN 包

如果用于电视的 AIR 应用程序使用的是本机扩展，则创建 AIRN 包而不是 AIR 包。若要创建 AIRN 包，请使用 ADT package 命令，同时将目标类型设置为 airn。

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp-app.xml myApp.swf
icons -extdir C:\extensions
```

此示例假设：

- 命令行 shell 的路径定义中已定义了 ADT 工具的路径。（请参阅第 237 页的 “ 路径环境变量 ”。）
- 证书 codesign.p12 位于从中运行 ADT 命令的父目录中。
- 参数 -extdir 命名包含应用程序使用的 ANE 文件的目录。

  这些 ANE 文件包含扩展的纯 ActionScript 存根或模拟器版本。包含本机代码的扩展版本安装在用于电视的 AIR 设备上。

从包含应用程序文件的目录运行此命令。示例中的应用程序文件是 myApp-app.xml （应用程序描述符文件）、 myApp.swf 和图标目录。

当运行如上所示的命令时， ADT 会提示输入 keystore 密码。并非所有的 shell 程序都会显示键入的密码字符；只需在键入结束后按 Enter 即可。此外，也可以使用 storepass 参数将密码包含在命令中。

还可以为使用本机扩展的用于电视的 AIR 应用程序创建 AIRI 文件。 AIRI 文件与 AIRN 文件相似，只不过该文件没有签名。例如：

```
adt -prepare myApp.airi myApp.xml myApp.swf icons -extdir C:\extensions
```

当您准备好对应用程序进行签名时，可以从 AIRI 文件创建 AIRN 文件：

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp.airi
```

**更多帮助主题**

针对 Adobe AIR 开发本机扩展

## 使用 Flash Builder 或 Flash Professional 打包

使用 Flash Professional 和 Flash Builder，您可以无需亲自运行 ADT 便可发布或导出 AIR 包。为 AIR 应用程序创建 AIR 包的过程包含在这些程序的文档中。

但是，目前只有 ADT 才能创建 AIRN 包 （使用本机扩展的用于电视的 AIR 应用程序的应用程序包）。

**更多帮助主题**

使用 Flash Builder 打包 AIR 应用程序

针对 Adobe AIR 进行发布

# 调试用于电视的 AIR 应用程序

## 使用 ADL 的设备模拟

用来测试和调试大多数应用程序功能的一种最快、最简单的方法是：使用 Adobe Debug Launcher (ADL) 实用程序在开发计算机上运行您的应用程序。

ADL 使用应用程序描述符中的 supportedProfiles 元素来选择要使用的配置文件。具体来说：

- 如果列出多个配置文件，则 ADL 会使用列表中的第一个配置文件。
- 可以使用 ADL 的 -profile 参数来选择 supportedProfiles 列表中的某个其他配置文件。
- 如果应用程序描述符中不包括 supportedProfiles 元素，则可以为 -profile 参数指定任何配置文件。

例如，使用以下命令启动应用程序以模拟 tv 配置文件：

```
adl -profile tv myApp-app.xml
```

使用 ADL 在桌面上模拟 tv 或 extendedTV 配置文件时，应用程序会在与目标设备更为相似的环境中运行。例如：

- 不属于 -profile 参数指定的配置文件一部分的 ActionScript API 不可用。
- ADL 允许通过菜单命令输入设备输入控件 （例如，遥控器）的输入指令。
- 通过在 -profile 参数中指定 tv 或 extendedTV， ADL 可以在桌面上模拟 StageVideo 类。
- 通过在 -profile 参数中指定 extendedTV，可让应用程序使用与应用程序 AIRN 文件一起打包的本机扩展存根或模拟器。

不过，由于 ADL 是在桌面上运行应用程序，因此，使用 ADL 测试用于电视的 AIR 应用程序具有以下限制：

- 不能反映应用程序在设备上的性能。请在目标设备上运行性能测试。
- 无法模拟 StageVideo 对象的限制。在面向用于电视的 AIR 设备时，您通常会使用 StageVideo 类而不是 Video 类来播放视频。 StageVideo 类可以利用设备硬件的性能优势，但存在显示限制。 ADL 在桌面上播放视频时不受这些限制的影响。因此，请在目标设备上测试视频播放。
- 无法模拟本机扩展的本机代码。但是，您可以在 ADL -profile 参数中指定 extendedTV 配置文件，该配置文件支持本机扩展。 ADL 允许您使用 ANE 包中包含的 扩展的纯 ActionScript 存根或模拟器版本进行测试。但是，设备上安装的相应 扩展通常也包括本机代码。若要使用带有本机代码的 扩展进行测试，请在目标设备上运行应用程序。

使用本机扩展

如果应用程序使用本机扩展，则 ADL 命令类似于以下示例：

```
adl -profile extendedTV -extdir C:\extensionDirs myApp-app.xml
```

此示例假设：

- 命令行 shell 的路径定义中已定义了 ADL 工具的路径。（请参阅第 237 页的 " 路径环境变量 "。）
- 当前目录包含应用程序文件。这些文件包括 SWF 文件和应用程序描述符文件，在此示例中，应用程序描述符文件为 myApp-app.xml。
- 参数 -extdir 命名一个目录，其中包含应用程序使用的各个本机扩展的目录。这些目录中的每个目录都包含本机扩展的未打包的 ANE 文件。例如：

```
C:\extensionDirs
    extension1.ane
        META-INF
            ANE
                default
                    library.swf
                extension.xml
            signatures.xml
        catalog.xml
        library.swf
        mimetype
    extension2.ane
        META-INF
            ANE
                default
                    library.swf
                extension.xml
            signatures.xml
        catalog.xml
        library.swf
        mimetype
```

这些未打包的 ANE 文件包含扩展的仅 ActionScript 存根或模拟器版本。包含本机代码的 扩展版本安装在用于电视的 AIR 设备上。

控件输入

ADL 可模拟 TV 设备上的远程控制按钮。当使用其中一个 TV 配置文件启动 ADL 时，您可以使用所显示的菜单将这些按钮输入发送到模拟设备上。

屏幕大小

您可以通过设置 ADL -screensize 参数在不同尺寸的屏幕上测试您的应用程序。可以指定包含四个值的字符串，这四个值分别表示正常屏幕和最大化屏幕的宽度和高度。例如：

```
adl -screensize 1024x728:1024x768 myApp-app.xml
```

**更多帮助主题**

第 121 页的 "AIR Debug Launcher (ADL)"

针对 Adobe AIR 开发本机扩展

## Trace 语句

在桌面上运行您的 TV 应用程序时，会将 trace 输出打印到调试器或用于启动 ADL 的终端窗口上。

## 使用 Flash Professional 进行远程调试

当用于电视的 AIR 应用程序在目标设备上运行时，可以使用 Flash Professional 对其进行远程调试。但是，设置远程调试的步骤取决于设备。例如，用于电视的 Adobe® AIR® MAX 2010 硬件开发工具包中包含有关该设备的详细步骤的文档。

但是，无论是何种目标设备，请执行以下步骤以准备远程调试：

1 在 " 发布设置 " 对话框的 "Flash" 选项卡中，选择 " 允许调试 "。

   通过选择此选项， Flash Professional 可在从 FLA 文件创建的所有 SWF 文件中包含调试信息。

2 在 "Adobe AIR 设置 " 对话框 （应用程序和安装程序设置）的 " 签名 " 选项卡中，选择用于准备 AIR 中间 (AIRI) 文件的选项。

   如果您仍在开发应用程序，使用不需要数字签名的 AIRI 文件就足够了。

3 发布您的应用程序，创建 AIRI 文件。

最后一个步骤是在目标设备上安装并运行应用程序。但是，这些步骤取决于设备。

## 使用 Flash Builder 进行远程调试

当用于电视的 AIR 应用程序在目标设备上运行时，还可以使用 Flash Builder 对其进行远程调试。但是，进行远程调试的步骤取决于设备。

但是，无论是何种目标设备，请执行以下步骤以准备远程调试：

1 选择 " 项目 ">" 导出发行版 "。选择用于准备 AIR 中间 (AIRI) 文件的选项。

   如果您仍在开发应用程序，使用不需要数字签名的 AIRI 文件就足够了。

2 发布您的应用程序，创建 AIRI 文件。

3 将应用程序的 AIRI 包更改为包含含有调试信息的 SWF 文件。

   包含调试信息的 SWF 文件位于应用程序的 Flash Builder 项目目录中的一个名为 bin-debug 的目录中。使用 bin-debug 目录中的 SWF 文件替换 AIRI 包中的 SWF 文件。

在 Windows 开发计算机上，可以通过执行以下操作来进行此替换：

1 对 AIRI 包文件进行重命名，使其具有文件扩展名 .zip 而不是 .airi。

2 解压缩 ZIP 文件内容。

3 在解压缩的目录结构中使用来自 bin-debug 的 SWF 文件替换 SWF 文件。

4 在解压缩的目录中重新压缩文件。

5 更改压缩文件，使其再次具有 .airi 文件扩展名。

如果使用的是 Mac 开发计算机，则此替换的步骤与设备有关。但是，通常包括以下步骤：

1 在目标设备上安装 AIRI 包。

2 使用 bin-debug 目录中的 SWF 文件替换目标设备上应用程序安装目录中的 SWF 文件。

   例如，以 Adobe AIR for TV MAX 2010 硬件开发工具包附带的设备为例。按照工具包文档中的说明安装 AIRI 包。然后，在 Mac 开发计算机命令行中，使用 telnet 访问目标设备。使用 bin-debug 目录中的 SWF 文件替换 /opt/adobe/stagecraft/apps/<application name>/ 下应用程序安装目录中的 SWF 文件。

以下是使用 Flash Builder 和 Adobe AIR for TV MAX 2010 硬件开发工具包附带的设备进行远程调试的步骤。

1 在运行 Flash Builder 的计算机以及开发计算机上，运行 MAX 2010 硬件开发工具包附带的用于电视的 AIR 设备连接器。该连接器会显示开发计算机的 IP 地址。

2 在硬件工具包设备上，启动开发工具包附带的 DevMaster 应用程序。

3 在 DevMaster 应用程序中，输入开发计算机的 IP 地址，如用于电视的 AIR 设备连接器中所示。

4 在 DevMaster 应用程序中，确保已选择 " 启用远程调试 "。

5 退出 DevMaster 应用程序。

6 在开发计算机上，选择 " 在用于电视的 AIR 连接器中启动 "。

7 在硬件工具包设备上，启动另一个应用程序。验证用于电视的 AIR 设备连接器中是否显示跟踪信息。

   如果未显示跟踪信息，则表示开发计算机和硬件开发包设备未连接。请确保开发计算机上用于跟踪信息的端口可用。可以在用于电视的 AIR 设备连接器中选择其他端口。此外，请确保防火墙允许访问所选择的端口。

然后，在 Flash Builder 中启动调试器。请执行以下操作：

1 在 Flash Builder 中，选择 " 运行 ">" 调试配置 "。

2 从用于本地调试的现有调试配置中复制项目名称。

3 在 " 调试配置 " 对话框中，选择 "Web 应用程序 "。然后选择 " 新建启动配置 " 图标。

4 将项目名称粘贴到 " 项目 " 字段中。

5 在 "URL 或启动路径 " 部分中，取消选中 " 使用默认值 "。也可以在文本字段中输入 **about:blank**。

6 选择 " 应用 " 以保存更改。

7 选择 " 调试 " 以启动 Flash Builder 调试器。

8 在硬件工具包设备上启动应用程序。

现在，您可以使用 Flash Builder 调试器设置断点和检查变量。

# 第 9 章：使用 Adobe AIR 的本机扩展

Adobe AIR 的本机扩展提供 ActionScript API，该 API 提供对采用本机代码编程的特定于设备的功能的访问。本机扩展开发人员有时与设备制造商合作，有时为第三方开发人员。

如果要开发本机扩展，请参阅针对 Adobe AIR 开发本机扩展。

本机扩展是以下项的组合：

- ActionScript 类。
- 本机代码。

不过，作为使用本机扩展的 AIR 应用程序开发人员，只使用 ActionScript 类。

在以下情况下，本机扩展非常有用：

- 本机代码实现提供对特定于平台的功能的访问。这些特定于平台的功能在内置 ActionScript 类中不可用，也无法在特定于应用程序的 ActionScript 类中实现。本机代码实现可以提供此类功能，因为它可以访问特定于设备的硬件和软件。
- 本机代码实现有时可能比仅使用 ActionScript 的实现速度更快。
- 本机代码实现可以提供对旧本机代码的 ActionScript 访问。

Adobe 开发人员中心提供了一些本机扩展的示例。例如，一个本机扩展提供对 Android 振动功能的 AIR 应用程序访问。请参阅 Adobe AIR 的本机扩展。

## AIR 本机扩展 (ANE) 文件

本机扩展开发人员将本机扩展打包到 ANE 文件中。 ANE 文件是一个归档文件，其中包含本机扩展所需的库和资源。

请注意，对于某些设备， ANE 文件包含本机扩展使用的本机代码库。但对于其他设备，设备上安装有本机代码库。在某些情况下，本机扩展对于某个特殊设备根本没有本机代码；其仅使用 ActionScript 实现。

作为 AIR 应用程序开发人员，采用如下方式使用 ANE 文件：

- 在应用程序的库路径中包含 ANE 文件，方法与在库路径中包含 SWC 文件相同。此操作允许应用程序引用扩展的 ActionScript 类。
- 使用 AIR 应用程序打包 ANE 文件。

## 本机扩展与 NativeProcess ActionScript 类

ActionScript 3.0 提供了一个 NativeProcess 类。此类允许 AIR 应用程序在主机操作系统上执行本机进程。此功能与本机扩展的功能类似，后者提供对特定于平台的功能和库的访问。在决定使用 NativeProcess 类还是使用本机扩展时，请考虑以下因素：

- 只有 extendedDesktop AIR 配置文件支持 NativeProcess 类。因此，对于具有 AIR 配置文件 extendedTV、mobileDevice 和 extendedMobileDevice 的应用程序，本机扩展是唯一的选择。
- 本机扩展开发人员通常为各种平台提供本机实现，但其提供的 ActionScript API 在各平台上通常相同。使用 NativeProcess 类时，不同平台上启动本机进程的 ActionScript 代码可能会不同。
- NativeProcess 类启动一个单独的进程，而本机扩展与 AIR 应用程序运行在同一进程中。因此，如果担心代码崩溃，则使用 NativeProcess 类比较安全。不过，单独的进程意味着可能需要实现进程间的通信处理。

# 本机扩展与 ActionScript 类库 （SWC 文件）

SWC 文件是采用归档格式的 ActionScript 类库。 SWC 文件包含 SWF 文件和其他资源文件。 SWC 文件是共享 ActionScript 类 （而不是共享各个 ActionScript 代码和资源文件）的简便方法。

本机扩展包是一个 ANE 文件。和 SWC 文件一样， ANE 文件也是一个 ActionScript 类库，包含归档格式的 SWF 文件和其他资源文件。不过， ANE 文件和 SWC 文件之间最重要的区别是只有 ANE 文件才能包含本机代码库。

**更多帮助主题**

关于 SWC 文件

# 支持的设备

从 AIR 3 开始，可以为以下设备在应用程序中使用本机扩展：

- Android 设备，从 Android 2.2 开始
- iOS 设备，从 iOS 4.0 开始
- Blackberry PlayBook
- 支持 AIR 3.0 的 Windows 桌面设备
- 支持 AIR 3.0 的 Mac OS X 桌面设备
- 用于电视的 AIR 设备，从用于电视的 AIR 2.5 开始

通常，同一本机扩展以多个平台为目标。扩展的 ANE 文件为每个支持的平台包含 ActionScript 和本机库。通常， ActionScript 库针对所有平台都具有相同的公共接口。本机库必须不同。

有时，本机扩展支持默认平台。默认平台的实现仅具有 ActionScript 代码，但无本机代码。如果为扩展不专门支持的平台打包应用程序，则应用程序在执行时使用默认实现。例如，假设有一个扩展，提供仅适用于移动设备的功能。该扩展还可以提供桌面应用程序可用于模拟功能的默认实现。

# 支持的设备配置文件

以下 AIR 配置文件支持本机扩展：

- extendedTV，从 AIR 2.5 开始
- extendedDesktop，从 AIR 3.0 开始
- mobileDevice，从 AIR 3.0 开始
- extendedMobileDevice，从 AIR 3.0 开始

**更多帮助主题**

AIR 配置文件支持

# 使用本机扩展的任务列表

要在应用程序中使用本机扩展，请执行以下任务：

1 在应用程序描述符文件中声明扩展。

2 在应用程序库路径中包含 ANE 文件。

3 打包应用程序 .

# 在应用程序描述符文件中声明扩展

所有 AIR 应用程序都具有应用程序描述符文件。当应用程序使用本机扩展时，应用程序描述符文件包含一个 <extensions> 元素。例如：

```
<extensions>
    <extensionID>com.example.Extension1</extensionID>
    <extensionID>com.example.Extension2<extensionID>
</extensions>
```

extensionID 元素的值与扩展描述符文件中的 id 元素的值相同。扩展描述符文件是一个名为 extension.xml 的 XML 文件。该文件打包在 ANE 文件中。您可使用归档提取器工具查看 extension.xml 文件。

# 在应用程序库路径中包含 ANE 文件

要编译使用本机扩展的应用程序，请在库路径中包含 ANE 文件。

## 使用带 Flash Builder 的 ANE 文件

如果应用程序使用本机扩展，请在库路径中包含本机扩展的 ANE 文件。然后可以使用 Flash Builder 来编译 ActionScript 代码。请执行以下操作：

1 在 Flash Builder 项目中，选择 " 项目 ">" 属性 "。

2 在 " 属性 " 对话框中选择 "ActionScript 生成路径 "。

3 在 " 库路径 " 选项卡中，选择 " 添加 SWC..."。

4 在 " 文件类型 " 下拉框中，选择 "*.*"。

5 浏览到 ANE 文件，然后选择 " 确定 "。

现在， ANE 文件将会显示在 " 属性 " 对话框的 " 库路径 " 选项卡中。

6 展开 ANE 文件条目。双击 " 链接类型 " 以打开 " 库路径项目选项 " 对话框。

7 在 " 库路径项目选项 " 对话框中，将 " 链接类型 " 更改为 " 外部 "。

现在，可以使用 " 项目 ">" 生成项目 " 来编译应用程序。

## 使用带 Flash Professional 的 ANE 文件

如果应用程序使用本机扩展，请在库路径中包含本机扩展的 ANE 文件。然后可以使用 Flash Professional CS5 来编译 ActionScript 代码。请执行以下操作：

1 将 ANE 文件的文件扩展名从 .ane 更改为 .swc。必须执行此步骤，以便 Flash Professional 可以找到文件。

2 在 FLA 文件中，选择“文件”>“ActionScript 设置”。

3 在“高级 ActionScript 3.0 设置”对话框中，选择“库路径”选项卡。

4 选择“浏览到 SWC 文件”按钮。

5 浏览到 SWC 文件，然后选择“打开”。

现在， SWC 文件将会显示在“高级 ActionScript 3.0 设置”对话框的“库路径”选项卡中。

6 选择 SWC 文件后，再选择“为库设置链接选项”按钮。

7 在“库路径项目选项”对话框中，将“链接类型”更改为“外部”。

# 打包使用本机扩展的应用程序

使用 ADT 可打包使用本机扩展的应用程序。无法打包使用 Flash Professional 或 Flash Builder 的应用程序。

有关使用 ADT 的详细信息，请参阅 AIR 开发人员工具 (ADT)。

例如，下面的 ADT 命令为使用本机扩展的应用程序创建一个 DMG 文件 （用于 Mac OS X 的本机安装程序文件）：

```
adt -package
    -storetype pkcs12
    -keystore myCert.pfx
    -target native
    myApp.dmg
    application.xml
    index.html resources
    -extdir extensionsDir
```

下面的命令为 Android 设备创建一个 APK 包：

```
adt -package
    -target apk
    -storetype pkcs12 -keystore ../codesign.p12
    myApp.apk
    myApp-app.xml
    myApp.swf icons
    -extdir extensionsDir
```

下面的命令为 iPhone 应用程序创建一个 iOS 包：

```
adt -package
    -target ipa-ad-hoc
    -storetype pkcs12 -keystore ../AppleDistribution.p12
    -provisioning-profile AppleDistribution.mobileprofile
    myApp.ipa
    myApp-app.xml
    myApp.swf icons Default.png
    -extdir extensionsDir
```

下面的命令创建一个 AIRN 包，此包是使用扩展的用于电视的 AIR 应用程序的包：

```
adt -package -storetype pkcs12 -keystore ../codesign.p12 -target airn myApp.airn myApp-app.xml myApp.swf
icons -extdir extensionsDir
```

请注意以下事项：

- 使用本机安装程序包类型。
- 指定扩展目录。
- 确保 ANE 文件支持应用程序的目标设备。

## 使用本机安装程序包类型

应用程序包必须是一个本机安装程序。您无法为使用本机扩展的应用程序创建跨平台 AIR 包 （.air 包），因为本机扩展通常包含本机代码。不过，通常本机扩展支持多个具有相同 ActionScript API 的本机平台。在这些情况下，可以在不同的本机安装程序包中使用同一 ANE 文件。

下表概述了用于 ADT 命令的 -target 选项的值：

| 应用程序的目标平台 | -target |
| --- | --- |
| Mac OS X 或 Windows 桌面设备 | -target native<br>-target bundle |
| 用于电视的 AIR 设备 | -target airn |
| Android | -target apk<br>或其他 Android 包目标。 |
| iOS | -target ipa-ad-hoc<br>或其他 iOS 包目标 |

## 指定扩展目录

使用 ADT 选项 -extdir 告知 ADT 包含本机扩展 （ANE 文件）的目录。

有关此选项的详细信息，请参阅第 137 页的 “文件和路径选项 ”。

## 确保 ANE 文件支持应用程序的目标设备

提供 ANE 文件时，本机扩展开发人员会告知扩展所支持的平台。您也可使用归档提取器工具查看 ANE 文件的内容。所提取的文件包含每个支持的平台的目录。

打包使用 ANE 文件的应用程序时，了解扩展所支持的平台十分重要。请考虑以下规则：

- 要创建 Android 应用程序包，ANE 文件必须包含 Android-ARM 平台。或者，ANE 文件必须包含默认平台且至少包含一个其他平台。
- 要创建 iOS 应用程序包，ANE 文件必须包含 iPhone-ARM 平台。或者，ANE 文件必须包含默认平台且至少包含一个其他平台。
- 要创建 Mac OS X 应用程序包，ANE 文件必须包含 MacOS-x86 平台。或者，ANE 文件必须包含默认平台且至少包含一个其他平台。
- 要创建 Windows 应用程序包，ANE 文件必须包含 Windows-x86 平台。或者，ANE 文件必须包含默认平台且至少包含一个其他平台。

注：用于电视的 AIR 设备使用设备捆绑的扩展。扩展的本机库安装在设备上。当应用程序在设备上运行时，用于电视的 AIR 将使用这些库。不过，用于电视的 AIR 扩展的 ANE 文件包含 default 平台和一个仅 ActionScript 实现。创建用于电视的 AIR 应用程序 （AIRN 文件）是 ANE 文件可以仅 具有默认平台的唯一情况。

# 第 10 章 : ActionScript 编译器

必须对 ActionScript 和 MXML 代码进行编译，才能将其包含在 AIR 应用程序中。如果使用 Adobe Flash Builder 或 Adobe Flash Professional 等集成开发环境 (IDE)，IDE 会在后台处理编译。但是，在未使用 IDE 时或在使用构建脚本时，也可以从命令行调用 ActionScript 编译器来创建 SWF 文件。

## 关于 Flex SDK 中的 AIR 命令行工具

用来创建 Adobe AIR 应用程序的每个命令行工具都会调用用于生成 Flex 应用程序的对应工具：

- amxmlc 调用 mxmlc 来编译应用程序类
- acompc 调用 compc 来编译库和组件类
- aasdoc 调用 asdoc，通过源代码注释生成文档文件

实用程序的 Flex 和 AIR 版本之间的唯一区别是，AIR 版本从 air-config.xml 文件加载配置选项，而不是从 flex-config.xml 文件加载。

Flex 文档库中的 《构建和部署 Flex 应用程序》详细介绍了 Flex SDK 工具和其命令行选项。此处仅对 Flex SDK 工具进行简要介绍，以便帮助您快速入门，并指出生成 Flex 应用程序与生成 AIR 应用程序之间的区别。

**更多帮助主题**

第 33 页的 " 使用 Flex SDK 创建第一个桌面 AIR 应用程序 "

## 编译器安装

通常在命令行中以及使用一个或多个配置文件来指定编译选项。 Flex SDK 全局配置文件包含编译器运行时所使用的默认值。可以编辑此文件以适合自己的开发环境。Flex SDK 安装的 frameworks 目录中包含两个 Flex 全局配置文件。运行 amxmlc 编译器时使用 air-config.xml 文件。此文件通过包括 AIR 库来为 AIR 配置编译器。运行 mxmlc 时使用 flex-config.xml 文件。

默认配置值适用于学习和了解 Flex 和 AIR 的工作方式，但是，如果要处理一个完整项目，则应更严格地检查可用选项。可以在本地配置文件中为编译器选项提供项目特定的值，对于给定项目来说，项目特定值优先于全局值。有关编译选项的完整列表以及这些配置文件的语法，请参阅 Flex 文档库的 《构建和部署 Flex 应用程序》 中的 "Flex SDK 配置 "。

注：没有专门用于 AIR 应用程序的编译选项，但在编译 AIR 应用程序时必须参考 AIR 库。通常情况下，这些库在项目级别的配置文件以及在生成工具 （例如 Ant）的对应文件中引用，或者在命令行中直接引用。

## 为 AIR 编译 MXML 和 ActionScript 源文件

可使用命令行 MXML 编译器 (amxmlc) 编译 AIR 应用程序的 Adobe® ActionScript® 3.0 和 MXML 资源。（您不需要编译基于 HTML 的应用程序。若要在 Flash Professional 中编译 SWF，只需将影片发布到 SWF 文件。）

使用 amxmlc 的基本命令行模式是：

```
amxmlc [compiler options] -- MyAIRApp.mxml
```

其中 [compiler options] 指定编译 AIR 应用程序所使用的命令行选项。

amxmlc 命令使用附加参数 +configname=air 调用标准 Flex mxmlc 编译器。该参数指示编译器使用 air-config.xml 文件，而不是 flex-config.xml 文件。在其他方面，使用 amxmlc 与使用 mxmlc 相同。

编译器加载的 air-config.xml 配置文件指定编译 AIR 应用程序通常所需的 AIR 和 Flex 库。您还可以使用本地项目级配置文件覆盖或添加全局配置的其他选项。通常，创建本地配置文件的最简便方式是编辑全局版本的副本。使用 -load-config 选项可以加载本地文件：

**-load-config=project-config.xml** 覆盖全局选项。

**-load-config+=project-config.xml** 向采用多个值的全局选项添加其他值，例如 -library-path 选项。仅采用一个值的全局选项将被覆盖。

如果对本地配置文件使用特定命名约定， amxmlc 编译器会自动加载本地文件。例如，如果主 MXML 文件为 RunningMan.mxml，则将本地配置文件命名为 RunningMan-config.xml。现在，您只需键入以下内容即可编译应用程序：

```
amxmlc RunningMan.mxml
```

由于 RunningMan-config.xml 的文件名与编译的 MXML 文件的文件名匹配，因此将自动加载该文件。

### amxmlc 示例

下面的示例演示如何使用 amxmlc 编译器。（只能编译应用程序的 ActionScript 和 MXML 资源。）

编译 AIR MXML 文件：

```
amxmlc myApp.mxml
```

编译和设置输出名称：

```
amxmlc -output anApp.swf -- myApp.mxml
```

编译 AIR ActionScript 文件：

```
amxmlc myApp.as
```

指定编译器配置文件：

```
amxmlc -load-config config.xml -- myApp.mxml
```

从其他配置文件添加其他选项：

```
amxmlc -load-config+=moreConfig.xml -- myApp.mxml
```

在命令行中添加库 （除了配置文件中的已有库以外）：

```
amxmlc -library-path+=/libs/libOne.swc,/libs/libTwo.swc  -- myApp.mxml
```

在不使用配置文件的情况下编译 AIR MXML 文件 (Win)：

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, ^
[AIR SDK]/frameworks/libs/air/airframework.swc, ^
-library-path [Flex 3 SDK]/frameworks/libs/framework.swc ^
-- myApp.mxml
```

在不使用配置文件的情况下编译 AIR MXML 文件 （Mac OS X 或 Linux）：

```
mxmlc -library-path [AIR SDK]/frameworks/libs/air/airframework.swc, \
[AIR SDK]/frameworks/libs/air/airframework.swc, \
-library-path [Flex 3 SDK]/frameworks/libs/framework.swc \
-- myApp.mxml
```

编译 AIR MXML 文件以便使用运行时共享库：

```
amxmlc -external-library-path+=../lib/myLib.swc -runtime-shared-libraries=myrsl.swf -- myApp.mxml
```

使用 Java 编译 （将类路径设置为包含 mxmlc.jar）：

```
java flex2.tools.Compiler +flexlib [Flex SDK 3]/frameworks +configname=air [additional compiler options] -
- myApp.mxml
```

flexlib 选项标识 Flex SDK 框架目录的位置，并使编译器能够查找 flex_config.xml 文件。

使用 Java 编译 （未设置类路径）:

```
java -jar [Flex SDK 2]/lib/mxmlc.jar +flexlib [Flex SDK 3]/frameworks +configname=air [additional compiler
options] -- myApp.mxml
```

使用 Apache Ant 调用编译器 （此示例使用 Java 任务运行 mxmlc.jar）:

```
<property name="SDK_HOME" value="C:/Flex3SDK"/>
<property name="MAIN_SOURCE_FILE" value="src/myApp.mxml"/>
<property name="DEBUG" value="true"/>
<target name="compile">
    <java jar="${MXMLC.JAR}" fork="true" failonerror="true">
        <arg value="-debug=${DEBUG}"/>
        <arg value="+flexlib=${SDK_HOME}/frameworks"/>
        <arg value="+configname=air"/>
        <arg value="-file-specs=${MAIN_SOURCE_FILE}"/>
    </java>
</target>
```

# 编译 AIR 组件或代码库 (Flex)

使用组件编译器 acompc 可以编译 AIR 库和独立的组件。除以下几点之外， acompc 组件编译器的行为与 amxmlc 编译器的行为类似:

- 必须指定库或组件应包含代码库的哪些类。
- acompc 不会自动查询本地配置文件。若要使用项目配置文件，必须使用 –load-config 选项。

acompc 命令调用标准 Flex compc 组件编译器，但该命令从 air-config.xml 文件而不是 flex-config.xml 文件加载其配置选项。

## 组件编译器配置文件

使用本地配置文件可以避免在命令行中键入 （甚至可能是错误键入）源路径和类名。向 acompc 命令行添加 -load-config 选项可以加载本地配置文件。

下面的示例演示如何使用 ParticleManager 和 Particle 两个类配置并构建库，这两个类均位于 com.adobe.samples.particles 包中。类文件位于 source/com/adobe/samples/particles 文件夹中。

```
<flex-config>
    <compiler>
        <source-path>
            <path-element>source</path-element>
        </source-path>
    </compiler>
    <include-classes>
        <class>com.adobe.samples.particles.ParticleManager</class>
        <class>com.adobe.samples.particles.Particle</class>
    </include-classes>
</flex-config>
```

若要使用配置文件编译名为 ParticleLib-config.xml 的库，请键入:

```
acompc -load-config ParticleLib-config.xml -output ParticleLib.swc
```

若要在命令行中完全运行同一命令，请键入:

```
acompc -source-path source -include-classes com.adobe.samples.particles.Particle
com.adobe.samples.particles.ParticleManager -output ParticleLib.swc
```

（在一行中键入整个命令，或者对命令解释程序使用续行符。）

## acompc 示例

这些示例假定您使用的是名为 myLib-config.xml 的配置文件。

编译 AIR 组件或库：

```
acompc -load-config myLib-config.xml -output lib/myLib.swc
```

编译运行时共享库：

```
acompc -load-config myLib-config.xml -directory -output lib
```

（请注意，在运行该命令之前，文件夹 lib 必须存在且必须为空。）

# 第 11 章 : AIR Debug Launcher (ADL)

使用 AIR Debug Launcher (ADL) 可以在开发期间运行基于 SWF 和基于 HTML 的应用程序。使用 ADL，您可以在不首先打包和安装应用程序的情况下运行该应用程序。默认情况下， ADL 使用 SDK 随附的运行时，这表示您无需单独安装运行时即可使用 ADL。

ADL 将 trace 语句和运行时错误输出到标准输出，但不支持断点或其他调试功能。可使用 Flash Debugger （或诸如 Flash Builder 或 Aptana Studio 的集成开发环境）解决复杂的调试问题。

AIR 支持直接进行调试，因此不需要运行时的调试版本 （就像使用 Adobe® Flash® Player 一样）。要执行命令行调试，请使用 Flash Debugger 和 AIR Debug Launcher (ADL)。

Flash Debugger 在 Flex SDK 目录中进行分发。本机版本 （例如， Windows 中的 fdb.exe）位于 bin 子目录中。 Java 版本位于 lib 子目录中。 AIR Debug Launcher adl.exe 位于 Flex SDK 安装的 bin 目录中。（没有单独的 Java 版本）。

注: 您不能无法使用 fdb 直接启动 AIR 应用程序, 因为 fdb 会尝试使用 Flash Player 来启动 AIR 应用程序。而是要让 AIR 应用程序连接到正在运行的 fdb 会话。

## ADL 用法

若要使用 ADL 运行应用程序，请使用以下模式:

```
adl application.xml
```

其中 application.xml 是应用程序的应用程序描述符文件。

ADL 的完整语法是:

```
adl [-runtime runtime-directory]
    [-pubid publisher-id]
    [-nodebug]
    [-atlogin]
    [-profile profileName]
    [-screensize value]
    [-extdir extension-directory]
    application.xml
    [root-directory]
    [-- arguments]
```

（中括号 [] 中的项目为可选项目。）

**-runtime runtime-directory** 指定包含要使用的运行时的目录。如果未指定，则使用 ADL 程序所在的相同 SDK 中的运行时目录。如果将 ADL 移到其 SDK 文件夹以外，则必须指定运行时目录。在 Windows 和 Linux 中，指定包含 Adobe AIR 目录的目录。在 Mac OS X 中，指定包含 Adobe AIR.framework 的目录。

**-pubid publisher-id** 分配指定值作为 AIR 应用程序的发行商 ID，以便完成此运行。通过指定临时的发行商 ID，您可以使用该发行商 ID 帮助唯一标识 AIR 应用程序，进而测试该应用程序的功能，例如通过本地连接进行通信。从 AIR 1.5.3 开始，还可以在应用程序描述符文件中指定发行商 ID （不应使用此参数）。

注：从 AIR 1.5.3 开始，不再自动计算发行商 ID，也不再自动向 AIR 应用程序分配发行商 ID。在创建对现有 AIR 应用程序的更新时，可以指定发行商 ID，但新应用程序不需要也不应指定发行商 ID。

**-nodebug** 关闭调试支持。如果使用此参数，应用程序进程则无法连接到 Flash Debugger，并禁止显示未处理异常对话框。（但是， trace 语句仍将输出到控制台窗口。）关闭调试可以使应用程序的运行速度略有提高，并可以更准确地模拟已安装的应用程序的执行模式。

**-atlogin** 在登录时模拟启动应用程序。此标志允许您测试应用程序逻辑是否仅在将应用程序设置为在用户登录时启动才可执行。使用 -atlogin 时，调度到应用程序的 InvokeEvent 对象的 reason 属性将是 login，而非 standard （除非应用程序正在运行）。

**-profile profileName** ADL 使用指定的配置文件对应用程序进行调试。 profileName 可以是以下值之一：

- 桌面
- extendedDesktop
- mobileDevice
- tv
- extendedTV

如果应用程序描述符包括 supportedProfiles 元素，则使用 -profile 指定的配置文件必须是受支持列表的成员。如果未使用 -profile 标记，则应用程序描述符中的第一个配置文件将用作活动配置文件。如果应用程序描述符不包括 supportedProfiles 元素，并且您未使用 -profile 标记，则会使用桌面配置文件。

有关详细信息，请参阅 第 180 页的 “supportedProfiles” 和第 186 页的 “ 设备配置文件 ”。

**-screensize value** 在桌面上运行 mobileDevice 配置文件中的应用程序时要使用的模拟屏幕大小。将屏幕大小指定为预定义类型，或者通过设置标准宽度和高度以及全屏宽度和高度的像素尺寸来指定屏幕大小。若要按类型指定值，请使用以下预定义屏幕类型之一：

| 屏幕类型 | 标准宽度 x 高度 | 全屏宽度 x 高度 |
|---|---|---|
| 480 | 720 x 480 | 720 x 480 |
| 720 | 1280 x 720 | 1280 x 720 |
| 1080 | 1920 x 1080 | 1920 x 1080 |
| Droid | 480 x 816 | 480 x 854 |
| FWQVGA | 240 x 432 | 240 x 432 |
| FWVGA | 480 x 854 | 480 x 854 |
| HVGA | 320 x 480 | 320 x 480 |
| iPad | 768 x 1004 | 768 x 1024 |
| iPhone | 320 x 460 | 320 x 480 |
| iPhoneRetina | 640 x 920 | 640 x 960 |
| iPod | 320 x 460 | 320 x 480 |
| NexusOne | 480 x 762 | 480 x 800 |
| QVGA | 240 x 320 | 240 x 320 |
| SamsungGalaxyS | 480 x 762 | 480 x 800 |
| SamsungGalaxyTab | 600 x 986 | 600 x 1024 |
| WQVGA | 240 x 400 | 240 x 400 |
| WVGA | 480 x 800 | 480 x 800 |

若要直接指定屏幕像素尺寸，请使用以下格式：

```
widthXheight:fullscreenWidthXfullscreenHeight
```

例如，可以按照以下方式来指定 NexusOne 屏幕：

```
-screensize 480x762:480x800
```

**-extdir extension-directory** 运行时应在其中搜索本机扩展的目录。目录包含应用程序使用的每个本机扩展的子目录。其中的每个子目录都包含扩展的未打包 ANE 文件。例如：

```
C:\extensionDirs\
    extension1.ane\
        META-INF\
            ANE\
                Android-ARM\
                    library.swf
                    extension1.jar
                extension.xml
            signatures.xml
        catalog.xml
        library.swf
        mimetype
    extension2.ane\
        META-INF\
            ANE\
                Android-ARM\
                    library.swf
                    extension2.jar
                extension.xml
            signatures.xml
        catalog.xml
        library.swf
        mimetype
```

使用 -extdir 参数时，请考虑以下事项：

- ADL 命令要求每一个指定目录都具有 .ane 文件扩展名。不过，“.ane” 后缀前面的文件名部分可以是任何有效的文件名。此部分不 必与应用程序描述符文件的 extensionID 元素的值匹配。
- 您可以多次指定 -extdir 参数。
- ADT 和 ADL 工具使用 -extdir 参数的方式有所不同。在 ADT 中，该参数指定一个包含 ANE 文件的目录。
- 也可使用环境变量 AIR_EXTENSION_PATH 指定扩展目录。请参阅 第 141 页的 “ADT 环境变量 ”。

**application.xml** 应用程序描述符文件。请参阅第 154 页的 “AIR 应用程序描述符文件 ”。应用程序描述符是 ADL 要求的唯一参数，并且在多数情况下是唯一需要的参数。

**root-directory** 指定要运行的应用程序的根目录。如果未指定，则使用应用程序描述符文件所在的目录。

**-- arguments** 在 “--” 之后显示的任何字符串均作为命令行参数传递到应用程序。

注：如果启动的 AIR 应用程序已在运行，则不会启动该应用程序的新实例，而是对正在运行的实例调度 invoke 事件。

# ADL 示例

在当前目录中运行应用程序：

```
adl myApp-app.xml
```

在当前目录的子目录中运行应用程序：

```
adl source/myApp-app.xml release
```

运行应用程序，并传入 “tick” 和 “tock” 两个命令行参数：

```
adl myApp-app.xml -- tick tock
```

使用特定运行时运行应用程序:

```
adl -runtime /AIRSDK/runtime myApp-app.xml
```

运行不支持调试的应用程序:

```
adl -nodebug myApp-app.xml
```

使用移动设备配置文件运行应用程序，并模拟 Nexus One 屏幕大小:

```
adl -profile mobileDevice -screensize NexusOne myMobileApp-app.xml
```

使用 Apache Ant 运行应用程序 （示例中显示的路径适用于 Windows）:

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
<property name="APP_ROOT" value="c:/dev/MyApp/bin-debug"/>
<property name="APP_DESCRIPTOR" value="${APP_ROOT}/myApp-app.xml"/>

<target name="test">
    <exec executable="${ADL}">
        <arg value="${APP_DESCRIPTOR}"/>
        <arg value="${APP_ROOT}"/>
    </exec>
</target>
```

# ADL 退出和错误代码

下表列出了 ADL 输出的退出代码:

| 退出代码 | 说明 |
|---|---|
| 0 | 启动成功。 ADL 在 AIR 应用程序退出后退出。 |
| 1 | 成功调用已在运行的 AIR 应用程序。 ADL 立即退出。 |
| 2 | 用法错误。提供给 ADL 的参数错误。 |
| 3 | 无法找到运行时。 |
| 4 | 无法启动运行时。通常，当应用程序中指定的版本与运行时版本不匹配时会发生此情况。 |
| 5 | 发生未知原因错误。 |
| 6 | 无法找到应用程序描述符文件。 |
| 7 | 应用程序描述符内容无效。此错误通常指示 XML 格式错误。 |
| 8 | 无法找到主应用程序内容文件 （在应用程序描述符文件的 <content> 元素中指定）。 |
| 9 | 主应用程序内容文件不是有效的 SWF 或 HTML 文件。 |
| 10 | 应用程序不支持使用 -profile 选项指定配置文件。 |
| 11 | 当前配置文件不支持 -screensize 参数。 |

# 第 12 章 : AIR Developer Tool (ADT)

AIR Developer Tool (ADT) 是用于开发 AIR 应用程序的多用途命令行工具。您可以使用 ADT 执行以下任务：

- 将 AIR 应用程序打包为 .air 安装文件
- 将 AIR 应用程序打包为本机安装程序。例如：在 Windows 上打包为 .exe 安装程序文件，在 iOS 上打包为 .ipa，或者在 Android 上打包为 .apk
- 将本机扩展打包为 AIR 本机扩展 (ANE) 文件
- 使用数字证书对 AIR 应用程序签名
- 更改 （迁移）用于应用程序更新的数字签名
- 创建自签名的数字代码签名证书
- 远程安装、启动和卸载移动设备上的应用程序
- 远程安装和卸载移动设备上的 AIR 运行时

ADT 是一个 Java 程序，它包含在 AIR SDK 中。必须具有 Java 1.5 或更高版本才能使用该程序。SDK 包括用于调用 ADT 的脚本文件。若要使用此脚本，必须在路径环境变量中包含 Java 程序的位置。如果路径环境变量中还列出了 AIR SDK bin 目录，您可以在命令行中键入带有适当参数的 adt，以调用 ADT。（如果您不知道如何设置路径环境变量，请参阅操作系统文档。如需进一步帮助，请参阅第 237 页的 " 路径环境变量 "，其中介绍了在大部分计算机系统上设置路径的过程。）

要使用 ADT，至少需要 2GB 的计算机内存。如果您的计算机内存低于此数量，在运行 ADT 时可能会出现内存不足的情况，特别是针对 iOS 打包应用程序时。

假设 Java 和 AIR SDK bin 目录都包含在路径变量中，您可以使用以下基本语法运行 ADT：

```
adt -command options
```

注：大多数集成的开发环境（包括 Adobe Flash Builder、Adobe Flash Professional 和 Aptana Studio），都可以为您进行 AIR 应用程序的打包和签名。如果您已使用此类开发环境，则通常不需要使用 ADT 执行这些常见任务。然而，您可能仍需要使用 ADT 作为命令行工具来执行集成开发环境不支持的功能。另外，您还可以使用 ADT 作为命令行工具作为自动执行的构建过程的一部分。

## ADT 命令

传送到 ADT 的第一个参数指定以下某个命令。

- -package — 将 AIR 应用程序或 AIR 本机扩展 (ANE) 打包。
- -prepare — 将 AIR 应用程序打包成中间文件 (AIRI)，但不签名。无法安装 AIRI 文件。
- -sign — 对使用 -prepare 命令生成的 AIRI 包签名。-prepare 和 -sign 命令允许在不同时间执行打包和签名。您也可以使用 -sign 命令对 ANE 包进行签名或重新签名。
- -migrate — 将迁移签名应用到已签名的 AIR 包，以允许您使用新的或更新的代码签名证书。
- -certificate — 创建自签名的数字代码签名证书。
- -checkstore — 验证是否可以访问 keystore 中的数字证书。
- -installApp — 在某个设备或设备仿真器上安装 AIR 应用程序。
- -launchApp — 在某个设备或设备仿真器上启动 AIR 应用程序。
- -appVersion — 报告目前在设备或设备仿真器上安装的 AIR 应用程序的版本。

- -uninstallApp — 卸载某个设备或设备仿真器上的 AIR 应用程序。
- -installRuntime — 在某个设备或设备仿真器上安装 AIR 运行时。
- -runtimeVersion — 报告在设备或设备仿真器上目前安装的 AIR 运行时的版本。
- -uninstallRuntime — 卸载某个设备或设备仿真器上目前安装的 AIR 运行时。
- -version — 报告 ADT 版本号。
- -help — 显示命令和选项的列表。

许多 ADT 命令共享相关的选项标志和参数的组合。对这些选项组合有单独的详细描述：


- 第 135 页的 “ADT 代码签名选项 ”
- 第 137 页的 “ 文件和路径选项 ”
- 第 137 页的 “ 调试器连接选项 ”
- 第 138 页的 “ 本机扩展选项 ”


## ADT package 命令

-package 命令应从主应用程序目录运行。该命令使用以下语法：

从组件应用程序文件创建 AIR 包：

```
adt -package
    AIR_SIGNING_OPTIONS
    -target packageType
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    FILE_OPTIONS
```

从组件应用程序文件创建本机包：

```
adt -package
    -target packageType
    DEBUGGER_CONNECTION_OPTIONS
    -airDownloadURL URL
    NATIVE_SIGNING_OPTIONS
    output
    app_descriptor
    -platformsdk path
    FILE_OPTIONS
```

从 AIR 或 AIRI 文件创建本机包：

```
adt -package
    -target packageType
    NATIVE_SIGNING_OPTIONS
    output
    input_package
```

从组件本机扩展文件创建本机扩展包：

注：不必对 ANE 文件进行签名，因此 AIR_SIGNING_OPTIONS 参数在本示例中是可选的。

```
adt -package
    AIR_SIGNING_OPTIONS
    -target ane
    output
    ANE_OPTIONS
```

**AIR_SIGNING_OPTIONS** AIR 签名选项标识用于对 AIR 安装文件进行签名的证书。第 135 页的 “ADT 代码签名选项 ” 详细介绍了各个签名选项。

**-target** 要创建的包的类型受支持的包类型包括：

- air — AIR 包。 air 是默认值，当创建 AIR 或 AIRI 文件时，不需要指定 -target 标志。
- airn — 用于扩展电视配置文件中的设备的本机应用程序包。
- ane — AIR 本机扩展包
- Android 包目标：
  - apk — Android 包。使用此目标生成的包只能安装在 Android 设备上，而不能安装在仿真器上。
  - apk-debug — 带有额外调试信息的 Android 包。（应用程序中的 SWF 文件也必须在具有调试支持的情况下进行编译。）
  - apk-emulator — 用于仿真器上不带调试支持的 Android 包。（使用 apk-debug 目标可允许同时在仿真器和设备上进行调试。）
  - apk-profile — 支持应用程序性能和内存分析的 Android 包。
- iOS 包目标：
  - ipa-ad-hoc — 用于临时分发的 iOS 包。
  - ipa-app-store — 用于 Apple 应用程序库分发的 iOS 包。
  - ipa-debug — 带有额外调试信息的 iOS 包。（应用程序中的 SWF 文件也必须在具有调试支持的情况下进行编译。）
  - ipa-test — 在不具有优化或调试信息的情况下编译的 iOS 包。
  - ipa-debug-interpreter — 在功能上等同于调试包，但编译速度更快。不过，它会解释 ActionScript 字节代码，但不会将其转换为机器代码。因此，解释程序包中的代码执行速度较慢。
  - ipa-test-interpreter — 在功能上等同于测试包，但编译速度更快。不过，它会解释 ActionScript 字节代码，但不会将其转换为机器代码。因此，解释程序包中的代码执行速度较慢。
- native — 本机桌面安装程序。生成的文件的类型是要运行命令的操作系统的本机安装格式：
  - EXE — Windows
  - DMG — Mac
  - DEB — Ubuntu Linux （AIR 2.6 或更早版本）
  - RPM — Fedora 或 OpenSuse Linux （AIR 2.6 或更早版本）

**DEBUGGER_CONNECTION_OPTIONS** 该调试器连接选项可以指定调试包是否应尝试连接到其他计算机上运行的远程调试器或侦听来自远程调试器的连接。该组选项仅支持移动调试包 （目标 apk-debug 和 ipa-debug）。第 137 页的 “ 调试器连接选项 ” 中对这些选项进行了说明。

**-airDownloadURL** 指定用于下载并在 Android 设备上安装 AIR 运行时的替代 URL。如果未指定，则在尚未安装该运行时的情况下， AIR 应用程序会将用户重定向到 Android Market 上的 AIR 运行时。

如果您的应用程序是通过替代 Market （Google 管理的 Android Market 以外的其他 Market）分发的，则您可能需要指定用于从该 Market 下载 AIR 运行时的 URL。某些替代 Market 不允许应用程序从该 Market 以外下载 AIR 运行时。此选项仅支持 Android 包。

**NATIVE_SIGNING_OPTIONS** 本机签名选项标识用于对本机包文件进行签名的证书。这些签名选项用于应用由本机操作系统而非 AIR 运行时所使用的签名。此外，这些选项与 AIR_SIGNING_OPTIONS 完全相同，第 135 页的 “ADT 代码签名选项 ” 对其进行了详细介绍。

Windows 和 Android 支持本机签名。在 Windows 上，应同时指定 AIR 签名选项和本机签名选项。在 Android 上，只能指定本机签名选项。

在许多情况下，您可以使用相同的代码签名证书来应用 AIR 和本机签名。但这并不是在所有情况下都适用。例如，对于提交至 Android Market 的应用程序，Google 的策略是：所有应用程序必须使用有效期至少至 2033 年的证书来签名。这表示不应将由知名证书颁发机构颁发的证书用于对 Android 应用程序进行签名（但在应用 AIR 签名时建议使用此类证书）。（任何证书颁发机构都不会颁发有效期如此之长的代码签名证书。）

**output** 要创建的包文件的名称。可以指定文件扩展名。如果没有指定，则会添加适合于 -target 值和当前操作系统的扩展名。

**app_descriptor** 指向应用程序描述符文件的路径。指定的路径可以是相对于当前目录的路径，也可以是绝对路径。（应用程序描述符文件在 AIR 文件中重命名为 application.xml。）

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**FILE_OPTIONS** 标识包中要包含的应用程序文件。第 137 页的“文件和路径选项”详细介绍了文件选项。当从 AIR 或 AIRI 文件创建本机包时，请不要指定文件选项。

**input_airi** 当从 AIRI 文件创建本机包时指定此项。如果目标是 air（或没有指定目标），则 AIR_SIGNING_OPTIONS 是必需的。

**input_air** 当从 AIR 文件创建本机包时指定此项。请不要指定 AIR_SIGNING_OPTIONS。

**ANE_OPTIONS** 标识用于创建本机扩展包的选项和文件。第 138 页的“本机扩展选项”中全面介绍了扩展包选项。

## ADT -package 命令示例

打包当前目录中针对基于 SWF 的 AIR 应用程序的特定应用程序文件：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf components.swc
```

打包当前目录中针对基于 HTML 的 AIR 应用程序的特定应用程序文件：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html AIRAliases.js image.gif
```

打包当前工作目录中的所有文件和子目录：

```
adt -package -storetype pkcs12 -keystore ../cert.p12 myApp.air myApp.xml .
```

注：keystore 文件包含用于对应用程序签名的私钥。切勿在 AIR 包中包含签名证书！如果在 ADT 命令中使用通配符，请将 keystore 文件放置到其他不同位置，使其不包含在包中。在本例中， keystore 文件 cert.p12 驻留在父目录中。

仅打包主文件和 images 子目录：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf images
```

打包基于 HTML 的应用程序和 HTML、 scripts 和 images 子目录中的所有文件：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml index.html AIRALiases.js html scripts
images
```

打包位于工作目录 (release/bin) 中的 application.xml 文件和主 SWF：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp.xml -C release/bin myApp.swf
```

对生成文件系统的多个位置的资源打包。在本例中，应用程序资源在打包前位于以下文件夹：

```
/devRoot
    /myApp
        /release
            /bin
                myApp-app.xml
                myApp.swf or myApp.html
    /artwork
        /myApp
            /images
                image-1.png
                ...
                image-n.png
    /libraries
        /release
            /libs
                lib-1.swf
                lib-2.swf
                lib-a.js
                AIRAliases.js
```

从 /devRoot/myApp 目录运行以下 ADT 命令：

```
adt -package -storetype pkcs12 -keystore cert.p12 myApp.air release/bin/myApp-app.xml
    -C release/bin myApp.swf (or myApp.html)
    -C ../artwork/myApp images
    -C ../libraries/release libs
```

生成以下包结构：

```
/myAppRoot
    /META-INF
        /AIR
            application.xml
            hash
    myApp.swf or myApp.html
    mimetype
    /images
        image-1.png
        ...
        image-n.png
    /libs
        lib-1.swf
        lib-2.swf
        lib-a.js
        AIRAliases.js
```

对于简单的基于 SWF 的应用程序，将 ADT 作为 Java 程序运行 （无需设置类路径）：

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf
```

对于简单的基于 HTML 的应用程序，将 ADT 作为 Java 程序运行 （无需设置类路径）：

```
java -jar {AIRSDK}/lib/ADT.jar -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.html
AIRAliases.js
```

作为 Java 程序运行 ADT （将 Java 类路径设置为包含 ADT.jar 包）：

```
java -com.adobe.air.ADT -package -storetype pkcs12 -keystore cert.p12 myApp.air myApp.xml myApp.swf
```

在 Apache Ant 中将 ADT 作为 Java 任务运行 （示例中显示的是用于 Windows 的路径）：

```
<property name="SDK_HOME" value="C:/AIRSDK"/>
<property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

target name="package">
    <java jar="${ADT.JAR}" fork="true" failonerror="true">
        <arg value="-package"/>
        <arg value="-storetype"/>
        <arg value="pkcs12"/>
        <arg value="-keystore"/>
        <arg value="../../ExampleCert.p12"/>
        <arg value="myApp.air"/>
        <arg value="myApp-app.xml"/>
        <arg value="myApp.swf"/>
        <arg value="icons/*.png"/>
    </java>
</target>
```

注：某些计算机系统可能会错误地解释文件系统路径中的双字节字符。如果发生此情况，请尝试将用于运行 ADT 的 JRE 设置为使用 UTF-8 字符集。用于在 Mac 和 Linux 上启动 ADT 的脚本默认采用此设置。在 Windows adt.bat 文件中，或当您直接从 Java 运行 ADT 时，请在 Java 命令行上指定 -Dfile.encoding=UTF-8 选项。

## ADT prepare 命令

-prepare 命令可创建未签名的 AIRI 包。 AIRI 包不能用于该包本身。使用 -sign 命令将 AIRI 文件转换为已签名的 AIR 包，或使用 package 命令将 AIRI 文件转换为本机包。

-prepare 命令使用以下语法：

```
adt -prepare output app_descriptor FILE_OPTIONS
```

**output** 创建的 AIRI 文件的名称。

**app_descriptor** 指向应用程序描述符文件的路径。指定的路径可以是相对于当前目录的路径，也可以是绝对路径。（应用程序描述符文件在 AIR 文件中重命名为 application.xml。）

**FILE_OPTIONS** 标识包中要包含的应用程序文件。第 137 页的 “ 文件和路径选项 ” 详细介绍了文件选项。

## ADT sign 命令

-sign 命令可对 AIRI 和 ANE 文件进行签名。

-sign 命令使用以下语法：

```
adt -sign AIR_SIGNING_OPTIONS input output
```

**AIR_SIGNING_OPTIONS** AIR 签名选项标识用于对包文件进行签名的证书。第 135 页的 “ADT 代码签名选项 ” 详细介绍了各个签名选项。

**input** 要签名的 AIRI 或 ANE 文件的名称。

**output** 要创建的已签名包的名称。

如果 ANE 文件已经签名，则将放弃旧签名。（AIR 文件不能重新签名 — 要使用新签名进行应用程序更新，请使用 -migrate 命令。）

## ADT migrate 命令

-migrate 命令将迁移签名应用到 AIR 文件。当您更新或更改数字证书，并需要更新使用旧证书签名的应用程序时，必须使用迁移签名。

注：必须在证书过期 180 天之内应用迁移证书。一旦超过此宽限期，就不能再使用迁移签名对应用程序更新进行签名。用户可以首先更新到使用迁移签名进行签名的应用程序版本，然后安装最新的更新；也可以先卸载原来的应用程序，然后安装新的 AIR 包。

若要使用迁移签名，请先使用新的或更新的证书对 AIR 应用程序签名 （使用 -package 或 -sign 命令），然后使用旧证书和 -migrate 命令应用迁移签名。

-migrate 命令使用以下语法：

```
adt -migrate AIR_SIGNING_OPTIONS input output
```

**AIR_SIGNING_OPTIONS** AIR 签名选项标识用于对 AIR 应用程序现有版本进行签名的原始证书。第 135 页的 "ADT 代码签名选项 " 详细介绍了各个签名选项。

**input** AIR 文件已使用新的应用程序证书进行签名。

**output** 最终包的名称，该包中包含来自新旧证书的签名。

注：为输入和输出 AIR 文件使用的文件名不得相同。

## ADT checkstore 命令

利用 -checkstore 命令可以检查 keystore 的有效性。该命令使用以下语法：

```
adt -checkstore SIGNING_OPTIONS
```

**SIGNING_OPTIONS** 标识要验证的 keystore 的签名选项。第 135 页的 "ADT 代码签名选项 " 详细介绍了各个签名选项。

## ADT certificate 命令

-certificate 命令允许您创建自签名的数字代码签名证书。该命令使用以下语法：

```
adt -certificate -cn name -ou orgUnit -o orgName -c country -validityPeriod years key-type output password
```

**-cn** 分配作为新证书公用名的字符串。

**-ou** 分配作为证书颁发组织单位的字符串。（可选。）

**-o** 分配作为证书颁发组织的字符串。（可选。）

**-c** 双字母 ISO-3166 国家 / 地区代码。如果提供的代码无效，则不会生成证书。（可选。）

**-validityPeriod** 证书有效的年限。如果未指定，则分配五年的有效期。（可选。）

**key_type** 用于证书的密钥类型，即 1024-RSA 或 2048-RSA。

**output** 要生成的证书文件的路径和文件名。

**password** 访问新证书所用的密码。当使用此证书对 AIR 文件签名时需要提供密码。

## ADT installApp 命令

-installApp 命令可在设备或仿真器上安装应用程序。

您必须先卸载现有的应用程序才能使用此命令进行重新安装。

该命令使用以下语法：

```
adt -installApp -platform platformName -platformsdk path-to-sdk -device deviceID -package fileName
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

**-package** 要安装的包的文件名。在 Android 上，此项必须是 APK 包。如果已安装指定的包， ADT 会返回错误代码 14：设备错误。

## ADT appVersion 命令

-appVersion 命令报告设备或仿真器上应用程序的已安装版本。该命令使用以下语法：

```
adt -appVersion -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

**-appid** 已安装应用程序的 AIR 应用程序 ID。如果在设备上未安装具有指定 ID 的应用程序， ADT 会返回退出代码 14：设备错误。

## ADT launchApp 命令

-launchApp 命令可在设备或仿真器上运行已安装的应用程序。该命令使用以下语法：

```
adt -launchApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

**-appid** 已安装应用程序的 AIR 应用程序 ID。如果在设备上未安装具有指定 ID 的应用程序， ADT 会返回退出代码 14：设备错误。

## ADT uninstallApp 命令

-uninstallApp 命令完全删除远程设备或仿真器上已安装的应用程序。该命令使用以下语法：

```
adt -uninstallApp -platform platformName -platformsdk path_to_sdk -device deviceID -appid applicationID
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

**-appid** 已安装应用程序的 AIR 应用程序 ID。如果在设备上未安装具有指定 ID 的应用程序， ADT 会返回退出代码 14：设备错误。

## ADT installRuntime 命令

-installRuntime 命令可在设备上安装 AIR 运行时。

您必须先卸载现有的 AIR 运行时版本才能使用此命令进行重新安装。

该命令使用以下语法：

```
adt -installRuntime -platform platformName -platformsdk path_to_sdk -device deviceID -package fileName
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

**-package** 要安装的运行时的文件名。在 Android 上，此项必须是 APK 包。如果未指定任何包，则会从 AIR SDK 的可用包中选择适用于设备或仿真器的适当运行时。如果已安装该运行时， ADT 会返回错误代码 14：设备错误。

## ADT runtimeVersion 命令

-runtimeVersion 命令报告设备或仿真器上 AIR 运行时的已安装版本。该命令使用以下语法：

```
adt -runtimeVersion -platform platformName -platformsdk path_to_sdk -device deviceID
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果没有安装运行时，或没有连接指定的设备， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

## ADT uninstallRuntime 命令

-uninstallRuntime 命令完全删除设备或仿真器上的 AIR 运行时。该命令使用以下语法：

```
adt -uninstallRuntime -platform platformName -platformsdk path_to_sdk -device deviceID
```

**-platform** 设备的平台的名称。目前只在 Android 平台上支持此命令。使用名称 android。

**-platformsdk** 指向目标设备的平台 SDK 的路径。目前，仅支持 Android 平台 SDK。 AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。此外，如果已设置了 AIR_ANDROID_SDK_HOME 环境变量，则不需要在命令行上提供平台 SDK 路径。（如果在两处都进行了设置，则会使用在命令行上提供的路径。）

**-device** 设备的序列号。只有当多个设备或仿真器连接到您的计算机并正在运行时，才需要指定设备。如果指定的设备未连接到计算机， ADT 会返回退出代码 14：设备错误。如果连接了多个设备或仿真器且没有指定某个设备， ADT 会返回退出代码 2：用法错误。

在 Android 上，使用 Android ADB 工具列出已连接的设备和运行中的仿真器的序列号：

```
adb devices
```

## ADT help 命令

ADT -help 命令显示命令行选项的简要提醒：

```
adt -help
```

help 命令输出中使用以下符号惯例：

- <> — 尖括号之间的项目表示必须提供的信息。
- () — 括号之间的项目表示在 help 命令输出中视为组合的选项。
- ALL_CAPS — 大写字母拼写的项目表示单独介绍的选项组合。
- | — 表示或者。例如，（ A | B ）表示项目 A 或项目 B。
- ? — 0 或 1。项目之后的问号表示项目是可选的，而且在使用该项目时仅出现一个实例。
- * — 0 或多个。项目之后的星号表示项目是可选的，而且可出现任何数目的实例。
- + — 1 或多个。项目之后的加号表示项目是必需的，而且可出现多个实例。
- 无符号 — 如果项目没有后缀符号，则该项目是必需的，且只能出现一个实例。

# ADT 选项组合

几个 ADT 命令共享公用的选项组合。

## ADT 代码签名选项

ADT 使用 Java 加密体系结构 (JCA) 访问对 AIR 应用程序签名所使用的私钥和证书。签名选项标识 keystore 以及该 keystore 中的私钥和证书。

keystore 必须包含私钥和关联的证书链。如果签名证书链接到某计算机上的受信任证书，则在 "AIR 安装 " 对话框中，证书公用名字段的内容会显示为发布者名称。

ADT 要求证书符合 x509v3 标准 (RFC3280)，并同时包含扩展密钥用法扩展和代码签名的相应值。应遵守证书中定义的约束，这些约束要求避免使用某些证书对 AIR 应用程序进行签名。

注：ADT 根据需要使用 Java 运行时环境代理设置来连接 Internet 资源，以便检查证书吊销列表和获取时间戳。如果在使用 ADT 连接这些 Internet 资源时遇到问题，并且网络需要特定的代理设置，则可能需要配置 JRE 代理设置。

### AIR 签名选项语法

签名选项使用以下语法 （在单个命令行上）：

```
-alias aliasName
-storetype type
-keystore path
-storepass password1
-keypass password2
-providerName className
-tsa url
```

**-alias** keystore 中的密钥的别名。当 keystore 仅包含一个证书时，则不必指定别名。如果未指定任何别名， ADT 则使用 keystore 中的第一个密钥。

并非所有 keystore 管理应用程序都允许向证书分配别名。例如，当使用 Windows 系统 keystore 时，则使用证书的识别名称作为别名。使用 Java Keytool 实用程序可以列出可用证书以便确定别名。例如，运行以下命令：

```
keytool -list -storetype Windows-MY
```

将为证书生成如下输出：

```
CN=TestingCert,OU=QE,O=Adobe,C=US, PrivateKeyEntry,
Certificate fingerprint (MD5): 73:D5:21:E9:8A:28:0A:AB:FD:1D:11:EA:BB:A7:55:88
```

若要在 ADT 命令行中引用此证书，请将别名设置为：

```
CN=TestingCert,OU=QE,O=Adobe,C=US
```

在 Mac OS X 中， Keychain 中的证书别名与在 Keychain Access 应用程序中显示的名称相同。

**-storetype** keystore 的类型，由 keystore 实现确定。大多数 Java 安装随附的默认 keystore 实现支持 JKS 和 PKCS12 类型。Java 5.0 包含对 PKCS11 类型和 Keychain 类型的支持，前者用于访问硬件标记中的 keystore，后者用于访问 Mac OS X keychain。 Java 6.0 包含对 MSCAPI 类型的支持 （在 Windows 中）。如果安装和配置了其他 JCA 提供程序，则可能还可以使用其他 keystore 类型。如果未指定任何 keystore 类型，则使用默认 JCA 提供程序的默认类型。

| 存储类型 | Keystore 格式 | 最低 Java 版本 |
|---|---|---|
| JKS | Java keystore 文件 (.keystore) | 1.2 |
| PKCS12 | PKCS12 文件 （.p12 或 .pfx） | 1.4 |

| 存储类型 | Keystore 格式 | 最低 Java 版本 |
|---|---|---|
| PKCS11 | 硬件标记 | 1.5 |
| KeychainStore | Mac OS X Keychain | 1.5 |
| Windows-MY 或 Windows-ROOT | MSCAPI | 1.6 |

**-keystore** 基于文件的存储类型的 keystore 文件路径。

**-storepass** 访问 keystore 所需的密码。如果未指定密码， ADT 将提示您输入密码。

**-keypass** 访问用于对 AIR 应用程序签名的私钥所需的密码。如果未指定密码， ADT 将提示您输入密码。

注：如果您输入密码作为 ADT 命令的一部分，则密码字符会保存在命令行历史记录中。因此，当证书的安全性很重要时，建议您不要使用 -keypass 或 -storepass 选项。另请注意，当忽略密码选项时，不会显示在密码提示处键入的字符 （由于相同的安全性原因）。只需键入密码并按 Enter 键即可。

**-providerName** 指定的 keystore 类型的 JCA 提供程序。如果未指定提供程序， ADT 将使用该 keystore 类型的默认提供程序。

**-tsa** 指定符合 RFC3161 的时间戳服务器的 URL，以便对数字签名创建时间戳。如果未指定任何 URL，则使用 Geotrust 提供的默认时间戳服务器。对 AIR 应用程序签名设置时间戳时，仍可以在签名证书过期之后安装该应用程序，这是因为时间戳验证该证书在签名时是否有效。

如果 ADT 无法连接到时间戳服务器，则取消签名，并且不会生成任何包。指定 -tsa none 可以禁用时间戳设置功能。但是，对于打包的没有时间戳的 AIR 应用程序，该应用程序将在签名证书过期后停止安装。

注: 许多签名选项与 Java Keytool 实用程序的相同选项具有同等的效用。您可以使用 Keytool 实用程序在 Windows 中检查和管理 keystore。也可以在 Mac OS X 上使用 Apple® 安全实用程序实现此目的。

**-provisioning-profile** Apple iOS 供给文件。（只有在打包 iOS 应用程序时才需要指定此参数。）

签名选项示例

使用 .p12 文件签名：

```
-storetype pkcs12 -keystore cert.p12
```

使用默认 Java keystore 签名：

```
-alias AIRcert -storetype jks
```

使用特定 Java keystore 签名：

```
-alias AIRcert -storetype jks -keystore certStore.keystore
```

使用 Mac OS X keychain 签名：

```
-alias AIRcert -storetype KeychainStore -providerName Apple
```

使用 Windows 系统 keystore 签名：

```
-alias cn=AIRCert -storeype Windows-MY
```

使用硬件标签签名 （请参考标记制造商提供的 Java 配置相关说明，以便使用该标记并获取正确的 providerName 值）：

```
-alias AIRCert -storetype pkcs11 -providerName tokenProviderName
```

在不嵌入时间戳的情况下签名：

```
-storetype pkcs12 -keystore cert.p12 -tsa none
```

## 文件和路径选项

文件和路径选项指定包中包含的所有文件。文件和路径选项使用以下语法：

```
files_and_dirs -C dir files_and_dirs -e file_or_dir dir -extdir dir
```

**files_and_dirs** 要在 AIR 文件中打包的文件和目录。可以指定任意数目的文件和目录，这些文件和目录以空格分隔。如果列出目录，则将该目录中的所有文件和子目录添加到该包中，但隐藏文件除外。（此外，如果指定应用程序描述符文件，则无论该文件是直接指定还是使用通配符或目录扩展指定的，都将忽略该文件，并且不会将其再次添加到包中。）指定的文件和目录必须位于当前目录或其子目录之一。使用 -C 选项可以更改当前目录。

重要说明：不能在 –C 选项后的 file_or_dir 参数中使用通配符。（命令解释程序先展开通配符，然后再将该参数传递到 ADT，这将导致 ADT 在错误位置中查找文件。）但是，您仍可以使用点字符 "." 表示当前目录。例如：-C assets。将资源目录中的所有内容 （包括任何子目录）都复制到应用程序包的根级别中。

**-C dir** 先将工作目录更改为 dir 的值，然后再处理添加到应用程序包中的后续文件和目录。将文件和目录添加到应用程序包的根目录中。 –C 选项可以使用任意次，以便包含文件系统多个点的文件。如果为 dir 指定相对路径，该路径则始终从原始工作目录解析。

由于 ADT 处理包包含的文件和目录，因此将存储当前目录和目标文件之间的相对路径。安装包时，这些路径将展开为应用程序目录结构。因此，指定 -C release/bin lib/feature.swf 会将 release/bin/lib/feature.swf 文件放置到根应用程序文件夹的 lib 子目录中。

**-e file_or_dir dir** 将文件或目录置于指定的包目录中。打包 ANE 文件时无法使用此选项。

注：应用程序描述符文件的 <content> 元素必须指定主应用程序文件在应用程序包目录树中的最终位置。

**-extdir dir*dir*** 的值是用于搜索本机扩展 （ANE 文件）的目录名称。指定一个绝对路径，或相对于当前目录的路径。您可多次指定 -extdir 选项。

指定的目录包含应用程序使用的本机扩展的 ANE 文件。此目录中的每个 ANE 文件都具有 .ane 文件名扩展。但是，.ane 文件扩展名前的文件名 不必 与应用程序描述符文件的 extensionID 元素的值相匹配。

例如，如果使用 -extdir ./extensions，则目录 extensions 可能如下所示：

```
extensions/
    extension1.ane
    extension2.ane
```

注：ADT 和 ADL 工具使用 -extdir 选项的方式有所不同。在 ADL 中，该选项指定一个包含子目录的目录，其中每个子目录都包含一个未打包的 ANE 文件。在 ADT 中，该选项指定一个包含 ANE 文件的目录。

## 调试器连接选项

当包的目标为 apk-debug 时，可以使用这些连接选项指定应用程序是会尝试连接到远程调试器还是侦听从远程调试器传入的连接。使用 -connect 选项可以连接到调试器；使用 -listen 选项可以通过 USB 连接接受来自调试器的连接。

-connect 选项使用以下语法：

```
-connect hostString
```

**-connect** 如果存在，应用程序将尝试连接到远程调试器。

**hostString** 标识运行 Flash 调试工具 FDB 的计算机的字符串。如果未指定，应用程序将尝试连接到创建包的计算机上所运行的调试器。主机字符串可以是完全限定的计算机域名：machinename.subgroup.example.com，也可以是 IP 地址：192.168.4.122。如果找不到指定的 （或默认的）机器，则运行时会显示对话框，请求有效的主机名称。

-listen 选项使用以下语法：

```
-listen port
```

**-listen** 如果存在，运行时会等待来自远程调试器的连接。

**port** （可选）要侦听的端口。默认情况下，运行时会侦听端口 7936。有关使用 -listen 选项的更多信息，请参阅第 83 页的 “通过 USB 使用 FDB 进行远程调试 ”。

## Android 应用程序分析选项

当包的目标是 apk-profile 时，可以使用分析器选项来指定要使用哪一个 SWF 文件进行性能和内存分析。分析器选项使用以下语法：

```
-preloadSWFPath directory
```

**-preloadSWFPath** 如果存在，则应用程序将尝试在指定的目录中查找预加载 SWF 文件。如果未指定，则 ADT 将包含 AIR SDK 中的预加载 SWF 文件。

**directory** 包含分析器预加载 SWF 文件的目录。

## 本机扩展选项

本机扩展选项指定用于为本机扩展打包 ANE 文件的选项和文件。这些选项与其中 -target 选项为 ane 的 ADT 包命令一起使用。

```
extension-descriptor -swc swcPath -platform platformName FILE_OPTIONS
```

**extension-descriptor** 本机扩展的描述符文件。

**-swc** 包含本机扩展的 ActionScript 代码和资源的 SWC 文件。

**-platform** 此 ANE 文件支持的平台的名称。可包含多个 -platform 选项，每个选项具有其自己的 FILE_OPTIONS。

**FILE_OPTIONS** 标识包中要包含的本机平台文件。第 137 页的 “ 文件和路径选项 ”详细介绍了文件选项。（请注意，在对 ANE 文件进行打包时不能使用 -e 选项。）

**更多帮助主题**

打包本机扩展

# ADT 错误消息

下表列出了 ADT 程序可能报告的错误以及可能的原因：

应用程序描述符验证错误

| 错误代码 | 说明 | 备注 |
|---|---|---|
| 100 | 无法分析应用程序描述符 | 检查应用程序描述符文件中是否有标签未封闭等 XML 语法错误。 |
| 101 | 缺少命名空间 | 添加缺少的命名空间。 |
| 102 | 命名空间无效 | 检查命名空间拼写。 |
| 103 | 意外的元素或属性 | 删除引起错误的元素和属性。描述符文件中不允许使用自定义值。<br>检查元素和属性名称的拼写。<br>确保将元素放置在正确的父元素内，且使用属性时对应着正确的元素。 |
| 104 | 缺少元素或属性 | 添加所需的元素或属性。 |

| 错误代码 | 说明 | 备注 |
| --- | --- | --- |
| 105 | 元素或属性所含的某个值无效 | 纠正引起错误的值。 |
| 106 | 窗口属性组合非法 | 某些窗口设置 （如 transparency = true 和 systemChrome = standard）不能在一起使用。更改其中某个不兼容的设置。 |
| 107 | 窗口最小大小大于窗口最大大小 | 更改最小大小或最大大小设置。 |
| 108 | 前面的元素中已使用的属性 | |
| 109 | 重复元素。 | 删除重复元素。 |
| 110 | 至少需要一个指定类型的元素。 | 添加缺少的元素。 |
| 111 | 在应用程序描述符中列出的配置文件都不支持本机扩展。 | 将配置文件添加到支持 本机扩展的 supportedProfies 列表。 |
| 112 | AIR 目标不支持本机扩展。 | 选择支持本机扩展的目标。 |
| 113 | <nativeLibrary> 和 <initializer> 必须一起提供。 | 必须为本机扩展中的每个本机库都指定初始值设定项函数。 |
| 114 | 找到不含 <nativeLibrary> 的 <finalizer>。 | 除非平台使用本机库，否则不要指定终结器。 |
| 115 | 默认平台不得包含本机实施。 | 请不要在默认平台元素中指定本机库。 |
| 116 | 此目标不支持浏览器调用。 | 对于指定的打包目标，<allowBrowserInvocation> 元素不能为 true。 |
| 117 | 此目标至少需要命名空间 n 打包本机扩展。 | 将应用程序描述符中的 AIR 命名空间更改为支持的值。 |

有关命名空间、元素、属性及其有效值的信息，请参阅第 154 页的“AIR 应用程序描述符文件”。

应用程序图标错误

| 错误代码 | 说明 | 备注 |
| --- | --- | --- |
| 200 | 无法打开图标文件 | 检查指定路径是否存在该文件。<br>使用另一个应用程序确保可以打开该文件。 |
| 201 | 图标大小错误 | 图标大小 （以像素为单位）必须与 XML 标签相匹配。例如，假设有应用程序描述符元素：<br><image32x32>icon.png</image32x32><br>icon.png 中的图像必须刚好为 32x32 像素。 |
| 202 | 图标文件包含的某种图像格式不受支持 | 仅支持 PNG 格式。将应用程序打包之前转换其他格式的图像。 |

应用程序文件错误

| 错误代码 | 说明 | 备注 |
| --- | --- | --- |
| 300 | 缺少文件，或无法打开文件 | 找不到或无法打开命令行中指定的文件。 |
| 301 | 缺少或无法打开应用程序描述符文件 | 在指定路径找不到应用程序描述符文件，或无法打开该文件。 |
| 302 | 包中缺少根内容文件 | 必须向包添加应用程序描述符的 <content> 元素中引用的 SWF 或 HTML 文件，方法是将这些文件加入到 ADT 命令行中列出的文件中。 |

| 错误代码 | 说明 | 备注 |
| --- | --- | --- |
| 303 | 包中缺少图标文件 | 必须向包添加应用程序描述符中指定的图标文件，方法是将这些图标加入到 ADT 命令行中列出的文件中。不会自动添加图标文件。 |
| 304 | 初始窗口内容无效 | 无法将应用程序描述符的 <content> 元素中引用的文件识别为有效的 HTML 或 SWF 文件。 |
| 305 | 初始窗口内容的 SWF 版本超出命名空间的版本 | 描述符命名空间中指定的 AIR 版本不支持应用程序描述符的 <content> 元素中所引用文件的 SWF 版本。例如，尝试将 SWF10 (Flash Player 10) 文件作为 AIR 1.1 应用程序的初始内容进行打包就会产生这种错误。 |
| 306 | 配置文件不受支持。 | 您在应用程序描述符文件中指定的配置文件不受支持。请参阅 第 180 页的 "supportedProfiles"。 |
| 307 | 命名空间必须至少为 nnn。 | 针对应用程序中使用的功能，使用适当的命名空间 （如 2.0 命名空间）。 |

其他错误的退出代码

| 退出代码 | 说明 | 备注 |
| --- | --- | --- |
| 2 | 用法错误 | 检查命令行参数是否存在错误 |
| 5 | 未知错误 | 此错误表示所发生的情况无法按常见的错误条件作出解释。可能的根源包括 ADT 与 Java 运行时环境之间不兼容、 ADT 或 JRE 安装损坏以及 ADT 内有编程错误。 |
| 6 | 无法写入输出目录 | 确保指定的 （或隐含的）输出目录可访问，并且所在驱动器有足够的磁盘空间。 |
| 7 | 无法访问证书 | 确保正确指定了密钥存储库的路径。<br>检查能否访问密钥存储库中的证书。可以使用 Java 1.6 Keytool 实用程序帮助排除证书访问权限方面的问题。 |
| 8 | 证书无效 | 证书文件格式错误、被修改、已到期或被撤消。 |
| 9 | 无法为 AIR 文件签名 | 验证传递给 ADT 的签名选项。 |
| 10 | 无法创建时间戳 | ADT 无法与时间戳服务器建立连接。如果通过代理服务器连接到 Internet，则可能需要配置 JRE 的代理服务器设置。 |
| 11 | 创建证书时出错 | 验证用于创建签名的命令行参数。 |
| 12 | 输入无效 | 验证命令行中传递给 ADT 的文件路径和其他参数。 |
| 13 | 缺少设备 SDK | 验证设备 SDK 配置。ADT 找不到执行指定命令所需的设备 SDK。 |
| 14 | 设备错误 | ADT 无法执行命令，因为存在设备限制或设备问题。例如，在尝试卸载未实际安装的应用程序时会显示此退出代码。 |

| 退出代码 | 说明 | 备注 |
| --- | --- | --- |
| 15 | 无设备 | 验证设备是否已连接且已开启，或仿真器是否正在运行。 |
| 16 | 缺少 GPL 组件 | 当前的 AIR SDK 未包含执行请求的操作所需的所有组件。 |
| 17 | 设备打包工具失败。 | 由于缺少预期的操作系统组件，因此无法创建包。 |

**Android** 错误

| 退出代码 | 说明 | 备注 |
| --- | --- | --- |
| 400 | 当前的 Android sdk 版本不支持属性。 | 检查属性名称的拼写是否正确，以及对于在其中出现的元素是否为有效的属性。如果此属性是在 Android 2.2 之后新增的，您可能需要在 ADT 命令中设置 -platformsdk 标志。 |
| 401 | 当前的 Android sdk 版本不支持属性值 | 检查属性值的拼写是否正确，以及对于该属性是否为有效的值。如果此属性值是在 Android 2.2 之后新增的，您可能需要在 ADT 命令中设置 -platformsdk 标志。 |
| 402 | 当前的 Android sdk 版本不支持 XML 标签 | 检查 XML 标签名称的拼写是否正确，以及是否为有效的 Android 清单文档元素。如果此元素是在 Android 2.2 之后新增的，您可能需要在 ADT 命令中设置 -platformsdk 标志。 |
| 403 | 不允许覆盖 Android 标签 | 应用程序正在尝试覆盖保留以供 AIR 使用的 Android 清单元素。请参见 第 64 页的 "Android 设置 "。 |
| 404 | 不允许覆盖 Android 属性 | 应用程序正在尝试覆盖保留以供 AIR 使用的 Android 清单属性。请参见 第 64 页的 "Android 设置 "。 |
| 405 | Android 标签 %1 必须是 manifestAdditions 标签中的第一个元素 | 将指定标签移动到所需位置。 |
| 406 | Android 标签 %2 的属性 %1 具有无效值 %3。 | 为该属性提供有效值。 |

# ADT 环境变量

ADT 读取以下环境变量的值 （如果已设置）：

**AIR_ANDROID_SDK_HOME** 指定 Android SDK 根目录（包含 tools 文件夹的目录）的路径。AIR 2.6 以上版本的 SDK 包含 Android SDK 中实现相关 ADT 命令所需的工具。只有在使用其他版本的 Android SDK 时才需要设置此值。运行 ADT 命令需要 -platformsdk 选项；但如果设置了此变量，则在运行 ADT 命令时，就不需要指定该选项。如果同时设置了此变量和命令行选项，则会使用在命令行上指定的路径。

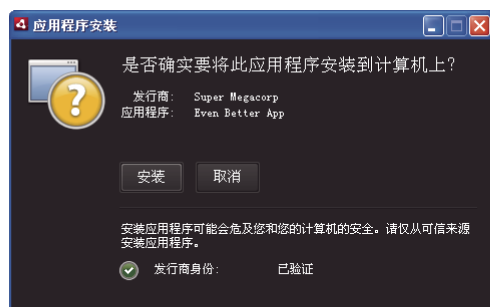
**AIR_EXTENSION_PATH** 指定用于搜索应用程序所需的本机扩展的目录列表。在搜索了 ADT 命令行上指定的任何本机扩展目录后，按顺序搜索此目录列表。 ADL 命令也会使用此环境变量。

注：某些计算机系统可能会错误地解释存储在这些环境变量中的双字节字符。如果发生此情况，请尝试将用于运行 ADT 的 JRE 设置为使用 UTF-8 字符集。用于在 Mac 和 Linux 上启动 ADT 的脚本默认采用此设置。在 Windows adt.bat 文件中，或当您直接从 Java 运行 ADT 时，请在 Java 命令行上指定 -Dfile.encoding=UTF-8 选项。

# 第 13 章：对 AIR 应用程序进行签名

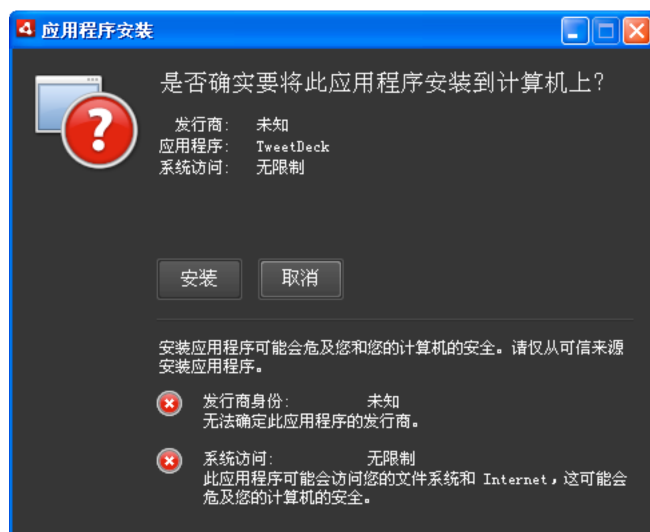
## 对 AIR 文件进行数字签名

如果用公认的认证机构 (CA) 颁发的证书对 AIR 安装文件进行数字签名，则可以为您的用户提供他们所安装的应用程序未经无意或恶意修改的重要保证，并能证明您的签名者 （发行商）身份。在使用受信任的证书或与安装计算机上受信任证书关联的证书对 AIR 安装程序进行签名后， AIR 在安装期间将显示发行商名称：




由受信任证书签名的应用程序的安装确认对话框

如果使用自签名证书 （或与受信任证书没有关联的证书）对应用程序进行签名，则用户在安装应用程序时必须承担更大的安全风险。安装对话框会反映这种额外风险：




由自签名证书签名的应用程序的安装确认对话框

重要说明：如果恶意实体以某种方式获取您的签名 keystore 文件或发现您的私钥，则可以以您的身份伪造 AIR 文件。

### 代码签名证书

证书实行声明 (CPS) 和由颁发证书的认证机构发布的订户协议中拟定了涉及代码签名证书使用的安全保证、限制和法律义务。有关当前颁发 AIR 代码签名证书的证书颁发机构的协议的详细信息，请参阅：

ChosenSecurity (http://www.chosensecurity.com/products/tc_publisher_id_adobe_air.htm)

ChosenSecurity CPS (http://www.chosensecurity.com/resource_center/repository.htm)

GlobalSign (http://www.globalsign.com/code-signing/index.html)

GlobalSign CPS (http://www.globalsign.com/repository/index.htm)

Thawte CPS (http://www.thawte.com/cps/index.html)

VeriSign CPS (http://www.verisign.com/repository/CPS/)

VeriSign Subscriber's Agreement (https://www.verisign.com/repository/subscriber/SUBAGR.html)

## 关于 AIR 代码签名

对 AIR 文件进行签名后，安装文件中将包含一个数字签名。此签名包括程序包的摘要，用于证实 AIR 文件自签名以来未经修改；此签名还包括有关签名证书的信息，用于证实发行商身份。

AIR 使用通过操作系统的证书存储区支持的公钥基础结构 (PKI) 来确定证书是否可信。安装 AIR 应用程序的计算机必须直接信任用于对此 AIR 应用程序进行签名的证书，或者必须信任将该证书链接到受信认证机构的证书链，才能核实发行商信息。

如果 AIR 文件用未链至其中一个受信根证书 （通常，这些证书包括所有自签名证书）的证书进行签名，则无法核实发行商信息。虽然 AIR 可以确定 AIR 程序包自签名以来未经修改，但无法知道文件的实际创建者和签名者。

注：用户可以选择信任自签名证书，这样，用该证书签名的任何 AIR 应用程序就会显示该证书中的公共名称字段的值作为发行商名称。 AIR 不为用户提供任何将证书指定为可信证书的方法。必须单独为用户提供证书 （不包括私钥），且用户必须使用操作系统提供的某种机制或适当的工具将证书导入系统证书存储区中的正确位置。

## 关于 AIR 发行商标识符

重要说明：从 AIR 1.5.3 开始，将弃用发行商 ID，并且不再基于代码签名证书计算发行商 ID。新应用程序不需要并且不应使用发行商 ID。更新现有应用程序时，必须在应用程序描述符文件中指定原始发行商 ID。

在 AIR 1.5.3 之前，AIR 应用程序安装程序在安装 AIR 文件的过程中生成发行商 ID。这是用于对 AIR 文件进行签名的证书的唯一标识符。如果对多个 AIR 应用程序重复使用同一个证书，它们将得到相同的发行商 ID。使用不同的证书 （有时甚至使用原始证书的续签实例）对应用程序更新进行签名都会更改发行商 ID。

在 AIR 1.5.3 和更高版本中，发行商 ID 不是由 AIR 分配的。使用 AIR 1.5.3 发布的应用程序可以在应用程序描述符中指定发行商 ID 字符串。只有针对最初为 AIR 1.5.3 之前的版本发布的应用程序发布更新时，才应指定发行商 ID。如果在应用程序描述符中没有指定原始 ID，则新 AIR 包不会被视为现有应用程序的更新。

要确定原始发行商 ID，请在安装原始应用程序的 META-INF/AIR 子目录中查找 publisherid 文件。此文件中的字符串就是发行商 ID。要手动指定发行商 ID，应用程序描述符必须在应用程序描述符文件的命名空间声明中指定 AIR 1.5.3 运行时 （或更高版本）。

发行商 ID （如果存在）用于以下用途：

- 作为加密本地存储加密密钥的一部分
- 作为应用程序存储目录路径的一部分
- 作为本地连接的连接字符串的一部分
- 作为用于使用 AIR 浏览器内 API 来调用应用程序的标识字符串的一部分
- 作为 OSID （在创建自定义安装 / 卸载程序时使用）的一部分

当发行商 ID 改变时，所有依赖该 ID 的 AIR 功能的行为也会改变。例如，将无法访问现有加密本地存储中的数据，所有创建到应用程序的本地连接的 Flash 或 AIR 实例必须使用连接字符串中的新 ID。在 AIR 1.5.3 或更高版本中，无法更改已安装应用程序的发行商 ID。如果在发布 AIR 包时使用不同的发行商 ID，安装程序会将新包视为不同的应用程序而不是更新。

## 关于证书格式

AIR 签名工具接受任何可通过 Java 加密体系结构 (JCA) 访问的 Keystore。这包括基于文件的 Keystore （例如 PKCS12 格式的文件，通常使用 .pfx 或 .p12 文件扩展名）、Java .keystore 文件、PKCS11 硬件 Keystore 和系统 Keystore。ADT 可以访问的 Keystore 格式取决于用于运行 ADT 的 Java 运行时的版本和配置。访问某些类型的 Keystore （例如 PKCS11 硬件令牌）可能需要安装和配置附加的软件驱动程序和 JCA 插件。

若要对 AIR 文件进行签名，可以使用大部分现有的代码签名证书，也可以获取一个专门为对 AIR 应用程序进行签名而颁发的新证书。例如，可以使用 VeriSign、 Thawte、 GlobalSign 或 ChosenSecurity 所颁发的以下任何类型的证书：

- ChosenSecurity
  - 用于 Adobe AIR 的 TC Publisher ID
- GlobalSign
  - ObjectSign 代码签名证书
- Thawte：
  - AIR 开发人员证书 (AIR Developer Certificate)
  - Apple 开发人员证书 (Apple Developer Certificate)
  - JavaSoft 开发人员证书 (JavaSoft Developer Certificate)
  - Microsoft 验证码证书 (Microsoft Authenticode Certificate)
- VeriSign：
  - Adobe AIR 数字 ID
  - Microsoft 验证码数字 ID (Microsoft Authenticode Digital ID)
  - Sun Java 签名数字 ID (Sun Java Signing Digital ID)

注：必须创建证书才能进行代码签名。不能使用 SSL 或其他证书类型对 AIR 文件进行签名。

## 时间戳

对 AIR 文件进行签名时，打包工具会查询时间戳机构的服务器，以获取可独立验证的签名日期和时间。获取的时间戳嵌入在 AIR 文件中。只要签名时签名证书有效，即使在证书过期后也可以安装 AIR 文件。另一方面，如果未获取时间戳，则在证书过期或被吊销之后， AIR 文件将变得不可安装。

默认情况下，AIR 打包工具会获取时间戳。然而，若要允许在时间戳服务不可用时打包应用程序，您可以禁用时间戳。Adobe 建议使所有公开分发的 AIR 文件都包含一个时间戳。

AIR 打包工具所采用的默认时间戳机构是 Geotrust。

## 获取证书

若要获取证书，您通常需要访问认证机构的网站，完成该公司的购买流程。使用何种工具生成 AIR 工具所需的 Keystore 文件，取决于所购买的证书的类型、证书在接收计算机上的存储方式，在某些情况下，还取决于用于获取证书的浏览器。例如，若要从 Thawte 获得和导出 Adobe Developer 证书，必须使用 Mozilla Firefox。然后可以直接从 Firefox 用户界面中以 .p12 或 .pfx 文件的形式导出证书。

注： Java 1.5 版及更高版本不允许在用于保护 PKCS12 证书文件的密码中使用高位 ASCII 字符。Java 由 AIR 开发工具用以创建签名的 AIR 包。将证书导出为 .p12 或 .pfx 文件时，只允许在密码中使用常规的 ASCII 字符。

可以使用用于打包 Air 安装文件的 Air 开发工具 (ADT) 生成自签名证书。也可以使用某些第三方工具。

有关如何生成自签名证书的说明以及有关对 AIR 文件进行签名的说明，请参阅 第 125 页的 “AIR Developer Tool (ADT)”。您还可以使用 Flash Builder、 Dreamweaver 和 AIR update for Flash 导出并对 AIR 文件进行签名。

下面的示例说明如何从 Thawte 认证机构获取 AIR 开发人员证书并准备好将它与 ADT 搭配使用。

## 示例：从 Thawte 获取 AIR 开发人员证书

注：有众多方法可用来获取和准备代码签名证书以供使用，此示例仅说明了其中的一种。每个证书颁发机构都有其自己的策略和程序。

若要购买 AIR 开发人员证书，Thawte 网站要求您使用 Mozilla Firefox 浏览器。此证书的私钥存储在浏览器的 Keystore 内。请确保 Firefox Keystore 受主密码保护并且计算机本身在物理上是安全的。（完成购买流程后，您就可以从浏览器 Keystore 中导出和删除证书及私钥。）

在证书注册过程中，将生成一个私钥 / 公钥对。私钥自动存储在 Firefox Keystore 内。从 Thawte 的网站请求和取回证书时，必须使用相同的计算机和浏览器。

1 访问 Thawte 网站，并浏览至代码签名证书的产品页。

2 从 "Code Signing Certificates" 列表中，选择 "Adobe AIR Developer Certificate"。

3 完成三步注册过程。您需要提供您所在单位的信息和联系信息。 Thawte 随后将执行其身份验证过程，并且可能要求提供其他信息。验证完成后， Thawte 将向您发送电子邮件，邮件中包含有关如何取回此证书的说明。

   注： 可以在此处找到有关所需文档类型的其他信息：https://www.thawte.com/ssl-digital-certificates/free-guides-whitepapers/pdf/enroll_codesign_eng.pdf。

4 从 Thawte 网站取回颁发的证书。证书会自动保存到 Firefox Keystore。

5 按照以下步骤从 Firefox Keystore 导出包含私钥和证书的 Keystore 文件：

   注：从 Firefox 导出私钥 / 证书时，它将以 ADT、 Flex、 Flash 和 Dreamweaver 可以使用的 .p12 (pfx) 格式导出。

   a 打开 Firefox 的 " 证书管理器 "(Certificate Manager) 对话框：

   b 在 Windows 中: 打开 "工具"(Tools) -> "选项"(Options) -> "高级"(Advanced) -> "加密"(Encryption) -> "查看证书"(View Certificates)

   c 在 Mac OS 中: 打开 "Firefox" -> "首选参数"(Preferences) -> "高级"(Advanced) -> "加密"(Encryption) -> "查看证书"(View Certificates)

   d 在 Linux 中: 打开 "编辑"(Edit) -> "首选参数"(Preferences) -> "高级"(Advanced) -> "加密"(Encryption) -> "查看证书"(View Certificates)

   e 从证书列表中选择 "Adobe AIR 代码签名证书"(Adobe AIR Code Signing Certificate)，然后单击 "备份"(Backup) 按钮。

   f 输入文件名和 Keystore 文件的导出位置，然后单击 " 保存 "(Save)。

   g 如果您使用的是 Firefox 主密码，系统将提示您输入软件安全设备的密码才能导出文件。（此密码仅由 Firefox 使用。）

   h 在 " 选择证书备份密码 "(Choose a Certificate Backup Password) 对话框中，为 Keystore 文件创建一个密码。

      重要说明：此密码用于保护 Keystore 文件，当使用该文件对 AIR 应用程序进行签名时需要提供此密码。应该选择一个安全密码。

   i 单击 " 确定 "。您应该会收到一条关于备份密码设置成功的消息。包含私钥和证书的 Keystore 文件以 .p12 文件扩展名保存 （采用 PKCS12 格式）。

6 通过 ADT、Flash Builder、Flash Professional 或 Dreamweaver 使用导出的 keystore 文件。只要对 AIR 应用程序进行签名，就需要提供为该文件创建的密码。

重要说明：私钥和证书仍存储在 Firefox Keystore 内。虽然这样使您可以导出证书文件的其他副本，但它同时也提供了另一个访问点，必须对此访问点加以保护才能维护证书和私钥的安全。

## 更改证书

在某些情况下，必须更改用于对 AIR 应用程序的更新进行签名的证书。此类情况包括：

- 续签原始签名证书。
- 从自签名证书升级到认证机构颁发的证书
- 从即将到期的自签名证书更改为另一个自签名证书
- 从一个商用证书更改为另一个商用证书，例如，当您的企业标识发生变化时

要使 AIR 将 AIR 文件识别为更新，必须使用同一证书对原始 AIR 文件和更新 AIR 文件进行签名，或者对更新应用证书迁移签名。迁移签名是使用原始证书对更新 AIR 包应用的第二次签名。迁移签名使用原始证书来证明签名者是应用程序的原始发行商。

安装了具有迁移签名的 AIR 文件后，新的证书将成为主要证书。后续更新不需要迁移签名。但是，您应尽可能地使用迁移签名，以便适应跳过更新的用户。

重要说明：您必须更改证书，并在原始证书过期前使用原始证书为更新应用迁移签名。否则，用户必须卸载应用程序的现有版本，才能安装新的版本。对于 AIR 1.5.3 或更高版本，您可以在证书到期后的 180 天宽限期内使用过期证书应用迁移签名。不过，您无法使用过期证书来应用主应用程序签名。

更改证书：

1 创建应用程序更新
2 将 AIR 更新文件打包并使用新证书对它进行签名
3 使用原始证书再次对此 AIR 文件进行签名 （使用 ADT -migrate 命令）

具有迁移签名的 AIR 文件在其他方面与普通 AIR 文件无异。如果应用程序安装在没有原始版本的系统中，则 AIR 会按照平常的安装方式安装新版本。

注：在 AIR 1.5.3 之前，使用续签证书对 AIR 应用程序进行签名时并不总是需要迁移签名。从 AIR 1.5.3 开始，续签证书将始终需要迁移签名。

第 130 页的 “ADT migrate 命令 ” 中介绍了应用迁移签名的过程。

### 应用程序标识更改

在 AIR 1.5.3 之前，安装使用迁移签名进行签名的更新时， AIR 应用程序的标识也会更改。更改应用程序的标识具有很多影响，包括：

- 新应用程序版本无法访问现有的加密本地存储区中的数据。
- 应用程序存储目录的位置会发生变化。旧位置中的数据不会复制到新目录。（但新应用程序可以根据旧发行商 ID 找到原始目录）。
- 应用程序无法再使用旧发行商 ID 打开本地连接。
- 用于从网页访问应用程序的标识字符串发生更改。
- 应用程序的 OSID 发生更改。（编写自定义安装 / 卸载程序时会使用 OSID）。

使用 AIR 1.5.3 发布时，无法更改应用程序标识。在更新 AIR 文件的应用程序描述符中必须指定原始应用程序和发行商 ID。否则，不会将新包识别为更新。

注：使用 AIR 1.5.3 或更高版本发布新的 AIR 应用程序时，不应指定发行商 ID。

## 术语

本节提供了一个术语表，阐释在决定如何对要公开发布的应用程序进行签名时应了解的部分关键术语。

| 术语 | 说明 |
| --- | --- |
| 认证机构 (CA) | 公钥基础结构网络中的一个实体，担当受信的第三方，并最终对公钥所有者的身份进行证实。 CA 通常会颁发由它自己的私钥进行签名的数字证书，以证明它已经核实了证书持有者的身份。 |
| 证书实行声明 (CPS) | 规定认证机构在颁发和核实证书方面的做法和政策。 CPS 是 CA 及其订户与信任方达成的合约的一部分。它还拟定了身份核实方面的政策及它们所提供的证书具备的保证程度。 |
| 证书吊销列表 (CRL) | 已被吊销而不应再受到信任的已颁发证书列表。 AIR 在对 AIR 应用程序进行签名时检查 CRL，如果不存在时间戳，它会在安装该应用程序时再次进行检查。 |
| 证书链 | 证书链是一个证书序列，链中的每个证书已由下一个证书进行签名。 |
| 数字证书 | 一种数字文档，包含所有者的身份、所有者的公钥以及证书本身的标识的有关信息。由认证机构颁发的证书本身由属于颁发证书的 CA 的证书进行签名。 |
| 数字签名 | 经过加密的消息或摘要，只能用公钥 - 私钥对的公钥部分解密。在 PKI 中，数字签名包含一个或多个最终来源于认证机构的数字证书。数字签名可用来证实消息 （或计算机文件）自签名以来未经修改 （在所用的加密算法提供的保证限制范围内）；此外，假如使用者信任颁发证书的认证机构，也可以用数字签名来证实签名者的身份。 |
| Keystore | 包含数字证书、在某些情况下也包含相关私钥的数据库。 |
| Java 加密体系结构 (JCA) | 一种用于管理和访问 Keystore 的可扩展体系结构。有关详细信息，请参阅 Java 加密体系结构参考指南。 |
| PKCS #11 | 由 RSA Laboratories 提出的加密令牌接口标准。也是一种基于硬件令牌的 Keystore。 |
| PKCS #12 | 由 RSA Laboratories 提出的个人信息交换语法标准。也是一种基于文件的 Keystore，通常包含私钥及其关联的数字证书。 |
| 私钥 | 由两部分组成的公钥 - 私钥非对称加密体系的私有部分。私钥必须保密，绝不应该通过网络传送。进行数字签名的消息由签名者通过私钥进行加密。 |
| 公钥 | 由两部分组成的公钥 - 私钥非对称加密体系的公开部分。公钥是公开提供的，用于解密用私钥加密的消息。 |
| 公钥基础结构 (PKI) | 认证机构用来证明公钥所有者身份的一种信任体系。网络客户端依靠受信的 CA 颁发的数字证书来核实数字消息 （或文件）签名者的身份。 |
| 时间戳 | 包含事件发生日期和时间的经过数字签名的数据。 ADT 可以将符合 RFC 3161 的时间服务器中的时间戳包含在 AIR 包中。如果存在时间戳， AIR 便在签名时使用时间戳确定证书的有效性。这样， AIR 应用程序便可在其签名证书过期后安装。 |
| 时间戳机构 | 颁发时间戳的机构。为了使 AIR 能够识别，时间戳必须符合 RFC 3161，并且时间戳签名必须链至安装计算机上的可信根证书。 |

## iOS 证书

Apple 颁发的代码签名证书用于对 iOS 应用程序进行签名，包括那些使用 Adobe AIR 开发的应用程序。要在测试设备上安装应用程序，必须使用 Apple 开发证书应用签名。要分发最终完成的应用程序，必须使用分发证书应用签名。

若要对应用程序进行签名， ADT 需要访问代码签名证书和关联的私钥。证书文件本身不包括私钥。您必须创建一个个人信息交换文件 （.p12 或 .pfx）形式的 keystore，该文件中同时包含证书和私钥。请参阅第 149 页的 “ 将开发人员证书转换为 P12 keystore 文件 ”。

### 生成证书签名请求

若要获取开发人员证书，应生成证书签名请求，然后在 Apple iOS Provisioning Portal 中提交该请求。

证书签名请求过程会生成一个公钥私钥对。私钥会保留在您的计算机上。您将包含公钥的签名请求以及您的标识信息发送给 Apple （该公司担任证书颁发机构）。 Apple 会利用自己的 World Wide Developer Relations 证书签署您的证书。

### 在 **Mac OS** 上生成证书签名请求

在 Mac OS 上，您可以使用钥匙串访问应用程序生成代码签名请求。钥匙串应用程序位于 " 应用程序 " 目录的 " 实用工具 " 子目录中。您可以在 Apple iOS Provisioning Portal 中查看有关生成证书签名请求的说明。

### 在 **Windows** 上生成证书签名请求

对于 Windows 开发人员，在 Mac 计算机上获取 iPhone 开发人员证书可能最容易。但是，也可以在 Windows 计算机上获取证书。首先，使用 OpenSSL 创建一个证书签名请求 （CSR 文件）：

1 将 OpenSSL 安装在 Windows 计算机上。（请访问 http://www.openssl.org/related/binaries.html。）

   您可能还需要安装 Open SSL 下载页上列出的 Visual C++ 2008 Redistributable 文件。（不需要在您计算机上安装 Visual C++。）

2 打开 Windows 命令会话，使用命令 CD 到 OpenSSL bin 目录 （例如 c:\OpenSSL\bin\）。

3 通过在命令行中输入下列内容创建私钥：

```
openssl genrsa -out mykey.key 2048
```

   保存此私钥文件。稍后您将使用它。

   当使用 OpenSSL 时，请不要忽略错误消息。即使 OpenSSL 生成错误消息，它也可能输出文件。不过，这些文件可能不可用。如果发现错误，请检查语法并再次运行命令。

4 通过在命令行中输入下列内容创建 CSR 文件：

```
openssl req -new -key mykey.key -out CertificateSigningRequest.certSigningRequest  -subj
"/emailAddress=yourAddress@example.com, CN=John Doe, C=US"
```

   使用您自己的电子邮件地址、 CN （证书名称）和 C （国家 / 地区）替换现有值。

5 将 CSR 文件上载到 Apple 的 iPhone 开发人员站点。（请参阅 " 申请 iPhone 开发人员证书并创建供给配置文件 "。）

## 将开发人员证书转换为 P12 keystore 文件

若要创建 P12 keystore，您必须将 Apple 开发人员证书和关联的私钥合并到一个文件中。创建 keystore 文件的过程取决于您用于生成原始证书签名请求的方法以及存储私钥的位置。

### 在 **Mac OS** 上将 **iPhone** 开发人员证书转换为 **P12** 文件

从 Apple 下载 Apple iPhone 证书后，将其导出为 P12 keystore 格式。在 Mac® OS 上执行以下操作：

1 打开钥匙串访问应用程序 （位于应用程序 / 实用工具文件夹中）。

2 如果尚未将该证书添加到钥匙串，请选择 " 文件 ">" 导入 "。然后浏览到您从 Apple 获取的证书文件 （.cer 文件）。

3 在钥匙串访问中选择密钥类别。

4 选择与 iPhone 开发证书相关联的私钥。

   该私钥由 iPhone 开发人员识别：与之配对的 < 名字 >< 姓氏 > 公共证书。

5 按住 Command 键单击 iPhone 开发人员证书，并选择 Export "iPhone Developer: Name..."。

6 以个人信息交换 (.p12) 文件格式保存您的 keystore。

7 系统将提示您创建一个密码，当您使用 keystore 对应用程序进行签名或者将在某个 keystore 中的密钥和证书传递到另一 keystore 时会用到该密码。

### 在 **Windows** 上将 **Apple** 开发人员证书转换为 **P12** 文件

若要开发 AIR for iOS 应用程序，必须使用 P12 证书文件。基于从 Apple 收到的 Apple iPhone 开发人员证书文件生成此证书。

1 将从 Apple 收到的开发人员证书文件转换成 PEM 证书文件。从 OpenSSL bin 目录运行以下命令行语句：

```
openssl x509 -in developer_identity.cer -inform DER -out developer_identity.pem -outform PEM
```

2 如果您使用的是 Mac 计算机上钥匙串中的私钥，则将其转换成 PEM 密钥：

```
openssl pkcs12 -nocerts -in mykey.p12 -out mykey.pem
```

3 现在，您可以基于密钥和 PEM 版本的 iPhone 开发人员证书生成有效的 P12 文件：

```
openssl pkcs12 -export -inkey mykey.key -in developer_identity.pem -out iphone_dev.p12
```

如果您使用的是 Mac OS 钥匙串中的密钥，则使用上一步骤中生成的 PEM 版本。否则，请使用以前生成的 OpenSSL 密钥（位于 Windows 上）。

# 使用 ADT 创建未签名的 AIR 中间文件

使用 -prepare 命令可以创建未签名的 AIR 中间文件。必须使用 ADT -sign 命令对 AIR 中间文件签名，以便生成有效的 AIR 安装文件。

-prepare 命令采用的标签和参数与 -package 命令相同 （但签名选项除外）。唯一区别在于前者的输出文件未签名。生成的中间文件的文件扩展名为：airi。

若要对 AIR 中间文件签名，请使用 ADT -sign 命令。（请参阅 第 130 页的 “ADT prepare 命令 ”。）

### ADT -prepare 命令示例

```
adt -prepare unsignedMyApp.airi myApp.xml myApp.swf components.swc
```

# 使用 ADT 对 AIR 中间文件进行签名

若要利用 ADT 对 AIR 中间文件签名，请使用 -sign 命令。sign 命令仅对 AIR 中间文件 （扩展名为 airi）起作用。不能对 AIR 文件进行两次签名。

若要创建 AIR 中间文件，请使用 adt -prepare 命令。（请参阅 第 130 页的 “ADT prepare 命令 ”。）

### 对 AIRI 文件签名

❖ 采用以下语法使用 ADT -sign 命令：

```
adt -sign SIGNING_OPTIONS airi_file air_file
```

**SIGNING_OPTIONS** 该签名选项标识用于对 AIR 文件签名的私钥和证书。第 135 页的 “ADT 代码签名选项 ” 介绍了这些选项。

**airi_file** 要进行签名的未签名 AIR 中间文件的路径。

**air_file** 要创建的 AIR 文件的名称。

### ADT -sign 命令示例

```
adt -sign -storetype pkcs12 -keystore cert.p12 unsignedMyApp.airi myApp.air
```

有关更多信息，请参阅 第 130 页的 “ADT sign 命令 ”。

# 对 AIR 应用程序的更新版本进行签名

您可以使用新证书或重续证书对现有 AIR 应用程序的更新版本进行签名。使用 ADT -migrate 命令将证书迁移签名应用于更新，如后面所述。

使用您的 AIR 应用程序的原始证书应用迁移签名。迁移签名会验证原始证书所有者是否已发布更新。

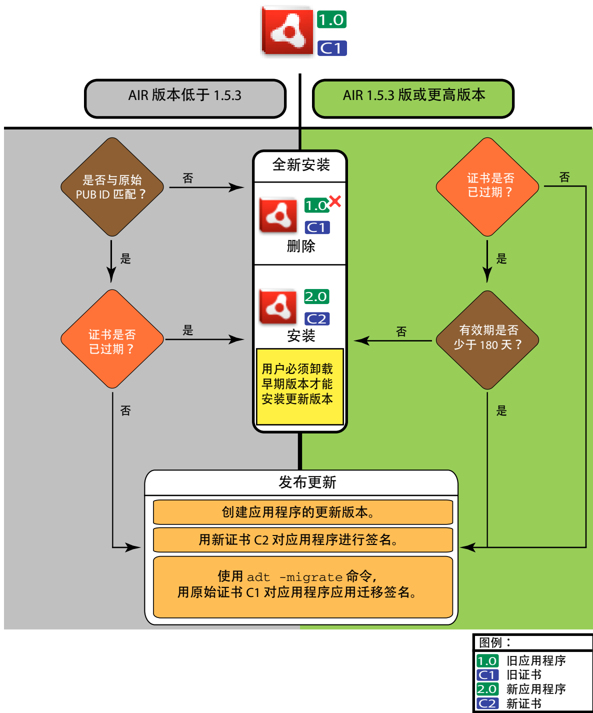
在应用迁移签名前，请考虑以下事项：

- 若要应用迁移签名，原始证书必须仍然有效或者是在过去的 180 天内过期的。此期间称为 " 宽限期 "，未来有可能变更该期间持续时间。
- 您不能在证书过期、并且已经超过 180 天宽限期后应用迁移签名。必须先卸载现有版本，才能安装更新版本。
- 180 天宽限期仅适用于在应用程序描述符命名空间中指定 AIR 1.5.3 版或更高版本的应用程序。

重要说明：利用过期证书中的迁移签名对更新进行签名是一个临时解决方案。若要创建一个全面解决方案，请创建一个标准化的签名工作流程来管理应用程序更新的部署。将具有最新证书的每个更新上载到单独的 URL，用户可以在此处部署应用程序。使用最新的证书在此部署 URL 处对每个新的更新进行签名，并且使用用于对上一更新进行签名的证书应用迁移。有关更多信息，请参阅第 198 页的 " 对应用程序更新工作流程进行签名 "。

下表总结了迁移签名的工作流程：

<table>
<tr><th>应用场景</th><th>证书 / 发布者 ID 的状态</th><th>开发人员操作</th><th>用户操作</th></tr>
<tr><td rowspan="4">• 基于 Adobe AIR 运行时版本 1.5.3 或更高版本的应用程序<br>• 证书 C1</td><td>证书的状态</td><td></td><td></td></tr>
<tr><td>有效</td><td>发布 AIR 应用程序的最新版本</td><td>无需任何操作<br>应用程序会自动升级</td></tr>
<tr><td>已过期并且超出宽限期</td><td>您不能将迁移签名应用于 AIR 应用程序更新。<br>而是必须使用新证书发布 AIR 应用程序的另一个版本。在卸载 AIR 应用程序的现有版本后，用户可以安装该程序的新版本。</td><td>卸载 AIR 应用程序的当前版本，然后安装最新版本</td></tr>
<tr><td>已过期，但在 180 天宽限期内</td><td>上述列中详细说明的操作之一</td><td>无需任何操作<br>应用程序会自动升级</td></tr>
<tr><td rowspan="6">• 基于 Adobe AIR 运行时版本 1.5.2 或更低版本的应用程序<br>• 证书 C1</td><td>应用程序描述符中的原始发布者 ID</td><td></td><td></td></tr>
<tr><td colspan="3">可用。与证书 C1 相关的下列详细信息适用：</td></tr>
<tr><td>有效</td><td>发布 AIR 应用程序的最新版本</td><td>无需任何操作<br>应用程序会自动升级</td></tr>
<tr><td>已过期并且超出宽限期</td><td>您不能将迁移签名应用于 AIR 应用程序更新。<br>而是必须使用新证书发布 AIR 应用程序的另一个版本。在卸载 AIR 应用程序的现有版本后，用户可以安装该程序的新版本。</td><td>卸载 AIR 应用程序的当前版本，然后安装最新版本</td></tr>
<tr><td colspan="3">不可用</td></tr>
<tr><td></td><td>发布 AIR 应用程序的最新版本</td><td>卸载 AIR 应用程序的当前版本，然后安装最新版本</td></tr>
</table>




适用于更新的签名工作流程

迁移应用程序：

1 创建应用程序更新

2 将 AIR 更新文件打包并使用新证书对它进行签名

3 使用原始证书和 -migrate 命令再次对 AIR 文件签名

使用 -migrate 命令签名的 AIR 文件可以用于安装应用程序的最新版本。该文件还可以用于更新使用旧证书签名的以前版本。

注：更新为早于 AIR 1.5.3 的版本发布的应用程序时，请在应用程序描述符中指定原始发布者 ID。否则，应用程序的用户必须先卸载早期版本才能安装更新版本。

### 迁移 AIR 应用程序以便使用新的证书

❖ 采用以下语法使用 ADT -migrate 命令：

```
adt -migrate SIGNING_OPTIONS air_file_in air_file_out
```

**SIGNING_OPTIONS** 该签名选项标识用于对 AIR 文件签名的私钥和证书。这些选项必须标识原始签名证书，第 135 页的“ADT 代码签名选项”介绍了这些选项。

**air_file_in** 使用新证书签名的要更新的 AIR 文件。

**air_file_out** 要创建的 AIR 文件。

注：用于输入和输出 AIR 文件的文件名必须是不同的。

### ADT 示例

```
adt -migrate -storetype pkcs12 -keystore cert.p12 myAppIn.air myApp.air
```

有关更多信息，请参阅 第 130 页的“ADT sign 命令”。

注：在 AIR 1.1 发行版中，已将 -migrate 命令添加到 ADT。

# 使用 ADT 创建自签名证书

您可以使用自签名证书生成有效的 AIR 安装文件。但是，自签名证书只为用户提供有限的安全保证。无法对自签名证书的真实性进行验证。当安装自签名 AIR 文件时，发行商信息将对用户显示为“未知”。 ADT 生成的证书有效期为五年。

如果对使用自签名证书签名的 AIR 应用程序创建更新，则必须使用相同证书对原始和更新 AIR 文件签名。即使使用相同的参数，ADT 生成的证书始终唯一。因此，如果希望使用 ADT 生成的证书对更新进行自签名，请将原始证书保存到安全位置。此外，在 ADT 生成的原始证书过期后，您无法生成更新的 AIR 文件。（您可以使用不同证书发布多个新应用程序，但是不能发布同一应用程序的多个新版本。）

重要说明: 鉴于自签名证书的限制，Adobe 强烈建议使用由信誉良好的证书颁发机构所颁发的商用证书对公开发行的 AIR 应用程序进行签名。

ADT 生成的证书和关联私钥存储在 PKCS12 类型的 keystore 文件中。指定的密码是针对密钥自身（而不是 keystore）设置的。

证书生成示例

```
adt -certificate -cn SelfSign -ou QE -o "Example, Co" -c US 2048-RSA newcert.p12 39#wnetx3tl
adt -certificate -cn ADigitalID 1024-RSA SigningCert.p12 39#wnetx3tl
```

若要使用这些证书对 AIR 文件签名，请使用 ADT -package 或 -prepare 命令的以下签名选项：

```
-storetype pkcs12 -keystore newcert.p12 -keypass 39#wnetx3tl
-storetype pkcs12 -keystore SigningCert.p12 -keypass 39#wnetx3tl
```

注：Java 1.5 版及更高版本不允许在用于保护 PKCS12 证书文件的密码中使用高位 ASCII 字符。只允许在密码中使用常规的 ASCII 字符。

# 第 14 章：AIR 应用程序描述符文件

每个 AIR 应用程序都需要一个应用程序描述符文件。应用程序描述符文件是定义应用程序基本属性的 XML 文档。

当您创建项目时，许多支持 AIR 的开发环境会自动生成应用程序描述符。如果不能自动生成，则您必须创建自己的描述符文件。示例描述符文件 descriptor-sample.xml 位于 AIR 和 Flex SDK 的 samples 目录中。

任何文件名都可用于应用程序描述符文件。当您打包应用程序时，应用程序描述符文件被重命名为 application.xml 并放置在 AIR 包内的特殊目录下。

示例应用程序描述符

以下应用程序描述符文档设置了大部分 AIR 应用程序都使用的基本属性：

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/3.0">
    <id>example.HelloWorld</id>
    <versionNumber>1.0.1</versionNumber>
    <filename>Hello World</filename>
    <name>Example Co. AIR Hello World</name>
     <description>
        <text xml:lang="en">This is an example.</text>
        <text xml:lang="fr">C'est un exemple.</text>
        <text xml:lang="es">Esto es un ejemplo.</text>
    </description>
    <copyright>Copyright (c) 2010 Example Co.</copyright>
    <initialWindow>
        <title>Hello World</title>
        <content>
            HelloWorld.swf
        </content>
    </initialWindow>
    <icon>
        <image16x16>icons/smallIcon.png</image16x16>
        <image32x32>icons/mediumIcon.png</image32x32>
        <image48x48>icons/bigIcon.png</image48x48>
        <image128x128>icons/biggerIcon.png</image128x128>
    </icon>
</application>
```

如果应用程序使用 HTML 文件而非 SWF 文件作为其根内容，不同的只是 <content> 元素：

```
<content>
    HelloWorld.html
</content>
```

## 应用程序描述符更改

以下 AIR 版本中的 AIR 应用程序描述符发生了更改。

### AIR 1.1 描述符更改

对于应用程序 name 和 description 元素，允许使用 text 元素进行本地化。

## AIR 1.5 描述符更改

contentType 成为了 fileType 的必需子项。

## AIR 1.5.3 描述符更改

添加了 publisherID 元素，使应用程序可以指定发行商 ID 值。

## AIR 2.0 描述符更改

添加了以下元素：

- aspectRatio
- autoOrients
- fullScreen
- image29x29
- image57x57
- image72x72
- image512x512
- iPhone
- renderMode
- supportedProfiles

## AIR 2.5 描述符更改

删除了以下元素：version

添加了以下元素：

- android
- extensionID
- extensions
- image36x36
- manifestAdditions
- versionLabel
- versionNumber

## AIR 2.6 描述符更改

添加了以下元素：

- image114x114
- requestedDisplayResolution
- softKeyboardBehavior

## AIR 3.0 描述符更改

添加了以下元素：

- colorDepth
- direct，作为 renderMode 的一个有效值

# 应用程序描述符文件结构

应用程序描述符文件是具有以下结构的 XML 文档：

```
<application xmlns="http://ns.adobe.com/air/application/3.0">
    <allowBrowserInvocation>...<allowBrowserInvocation>
    <android>
        <colorDepth>...</colorDepth>
        <manifestAdditions
                <manifest>...</manifest>
            ]]>
        </manifestAdditions
    </android>
    <copyright>...</copyright>
    customUpdateUI>...</
    <description>
        <text xml:lang="...">...</text>
    </description>
    <extensions>
        <extensionID>...</extensionID>
    </extensions>
    <filename>...</filename>
    <fileTypes>
        <fileType>
            <contentType>...</contentType>
            <description>...</description>
            <extension>...</extension>
            <icon>
                <imageNxN>...</imageNxN>
            </icon>
            <name>...</name>
        </fileType>
    </fileTypes>
    <icon>
        <imageNxN>...</imageNxN>
    </icon>
    <id>...</id>
    <initialWindow>
        <aspectRatio>...</aspectRatio>
        <autoOrients>...</autoOrients>
        <content>...</content>
        <fullScreen>...</fullScreen>
        <height>...</height>
        <maximizable>...</maximizable>
        <maxSize>...</maxSize>
        <minimizable>...</minimizable>
        <minSize>...</minSize>
        <renderMode>...</renderMode>
        <resizable>...</resizable>
        <systemChrome>...</systemChrome>
        <title>...</title>
        <transparent>...</transparent>
```

```
        <visible>...</visible>
        <width>...</width>
        <x>...</x>
        <y>...</y>
    </initialWindow>
    <installFolder>...</installFolder>
    <iPhone>
        <InfoAdditions>...</InfoAdditions>
        <requestedDisplayResolution>...</requestedDisplayResolution>
    </iPhone>
    <name>
        <text xml:lang="...">...</text>
    </name>
    <programMenuFolder>...</programMenuFolder>
    <publisherID>...</publisherID>
    <softKeyboardBehavior>...</softKeyboardBehavior>
    <supportedProfiles>...</supportedProfiles>
    <versionNumber>...</versionNumber>
    <versionLabel>...</versionLabel>
</application>
```

# AIR 应用程序描述符元素

下面的元素列表描述了 AIR 应用程序描述符文件的各个合法元素。

## allowBrowserInvocation

**Adobe AIR 1.0** 和更新版本 — 可选

启用 AIR 浏览器内置 API 来检测和启动应用程序。

如果将该值设置为 true，请务必考虑安全隐患。从浏览器调用 AIR 应用程序（针对 ActionScript 开发人员）和从浏览器调用 AIR 应用程序（针对 HTML 开发人员）中将对这些进行介绍。

有关更多信息，请参阅第 195 页的“从浏览器启动安装的 AIR 应用程序”。

父元素：第 158 页的“application”

子元素：无

内容
true 或 false （默认）

示例

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

## android

**Adobe AIR 2.5** 和更新版本 — 可选

可以将元素添加到 Android 清单文件。 AIR 为每个 APK 包创建了 AndroidManifest.xml 文件。您可以使用 AIR 应用程序描述符中的 android 元素将其他项目添加到其中。除 Android 外，忽略所有平台。

父元素：第 158 页的“application”

子元素：

- 第 161 页的 “colorDepth”
- 第 174 页的 “manifestAdditions”

内容

用于定义特定于 Android 的属性以添加到 Android 应用程序清单的元素。

示例

```
<android>
    <manifestAdditions>
        ...
    </manifestAdditions>
</android>
```

**更多帮助主题**

第 64 页的 “Android 设置 ”

AndroidManifest.xml 文件

## application

**Adobe AIR 1.0** 和更新版本 — 必需

AIR 应用程序描述符文档的根元素。

父元素：无

子元素：

- 第 157 页的 “allowBrowserInvocation”
- 第 157 页的 “android”
- 第 163 页的 “copyright”
- 第 163 页的 “customUpdateUI”
- 第 163 页的 “description”
- 第 165 页的 “extensions”
- 第 165 页的 “filename”
- 第 167 页的 “fileTypes”
- 第 168 页的 “icon”
- 第 169 页的 “id”
- 第 171 页的 “initialWindow”
- 第 172 页的 “installFolder”
- 第 173 页的 “iPhone”
- 第 176 页的 “name”
- 第 177 页的 “programMenuFolder”
- 第 178 页的 “publisherID”
- 第 179 页的 “softKeyboardBehavior”

- 第 180 页的 “supportedProfiles”
- 第 182 页的 “version”
- 第 183 页的 “versionLabel”
- 第 183 页的 “versionNumber”

属性

minimumPatchLevel — 此应用程序必需的 AIR 运行时最低修补级别。

xmlns — XML 命名空间属性决定应用程序所需的 AIR 运行时版本。

该命名空间因每个 AIR 主版本而异 （但不会因次要修补程序而异）。命名空间的最后一段 （如 “2.7,”）指示应用程序所需的运行时版本。

针对 AIR 版本的 xmlns 值为：

```
xmlns="http://ns.adobe.com/air/application/1.0"
xmlns="http://ns.adobe.com/air/application/1.1"
xmlns="http://ns.adobe.com/air/application/1.5"
xmlns="http://ns.adobe.com/air/application/1.5.2"
xmlns="http://ns.adobe.com/air/application/1.5.3"
xmlns="http://ns.adobe.com/air/application/2.0"
xmlns="http://ns.adobe.com/air/application/2.5"
xmlns="http://ns.adobe.com/air/application/2.6"
xmlns="http://ns.adobe.com/air/application/2.7"
```

对基于 SWF 的应用程序，应用程序描述符中指定的 AIR 运行时版本决定了可以作为应用程序初始内容加载的 SWF 最高版本。指定 AIR 1.0 或 AIR 1.1 的应用程序只能使用 SWF9 (Flash Player 9) 文件作为初始内容。即使运行应用程序时采用了 AIR 2 运行时也是如此。指定 AIR 1.5（或更高版本）的应用程序可以使用 SWF9 或 SWF10 (Flash Player 10) 文件作为初始内容。

SWF 版本决定了 AIR 和 Flash Player API 的哪个版本可供使用。如果将 SWF9 文件用作 AIR 1.5 应用程序的初始内容，则应用程序将只能访问 AIR 1.1 和 Flash Player 9 API。此外，AIR 2.0 或 Flash Player 10.1 中对现有 API 行为的更改将不会生效。（此原则有一个例外，在运行时当前或今后的修补程序中，可以追溯应用对 API 在安全方面的重要更改。）

对于基于 HTML 的应用程序，在应用程序描述符中指定的运行时版本确定了可供该应用程序使用的 AIR 和 Flash Player API 版本。HTML、CSS 和 JavaScript 的行为始终由已安装 AIR 运行时中所用 Webkit 版本决定，而非由应用程序描述符决定。

AIR 应用程序加载 SWF 内容时，可供该内容使用的 AIR 和 Flash Player API 的版本取决于内容的加载方式。有效版本有时取决于应用程序描述符命名空间，有时取决于正在加载内容的版本，有时取决于已加载内容的版本。下表展示了如何根据加载方法决定 API 版本：

| 加载内容的方式 | 决定 API 版本的方式 |
| --- | --- |
| 初始内容，基于 SWF 的应用程序 | 已加载文件的 SWF 版本 |
| 初始内容，基于 HTML 的应用程序 | 应用程序描述符命名空间 |
| 由 SWF 内容加载的 SWF | 正在加载的内容的版本 |
| 由 HTML 内容使用 <script> 标签加载的 SWF 库 | 应用程序描述符命名空间 |
| 由 HTML 内容使用 AIR 或 Flash Player API（如 flash.display.Loader）加载的 SWF | 应用程序描述符命名空间 |
| 由 HTML 内容使用 <object> 或 <embed> 标签（或等效的 JavaScript API）加载的 SWF | 已加载文件的 SWF 版本 |

当正在加载的 SWF 文件与正在加载的内容具有不同的版本时，可能会遇到两种问题：

- 早期版本的 SWF 加载更新版本的 SWF — 将取消解析所加载内容中对更新版本的 AIR 和 Flash Player 中添加的 API 的引用
- 更新版本的 SWF 加载早期版本的 SWF — 更新版本的 AIR 和 Flash Player 中更改的 API 可能不会按所加载内容的预期方式工作。

内容

应用程序元素包含定义 AIR 应用程序属性的子元素。

示例

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/2.7">
    <id>HelloWorld</id>
    <version>2.0</version>
    <filename>Hello World</filename>
    <name>Example Co. AIR Hello World</name>
     <description>
        <text xml:lang="en">This is an example.</text>
        <text xml:lang="fr">C'est un exemple.</text>
        <text xml:lang="es">Esto es un ejemplo.</text>
    </description>
    <copyright>Copyright (c) 2010 Example Co.</copyright>
    <initialWindow>
        <title>Hello World</title>
        <content>
            HelloWorld.swf
        </content>
        <systemChrome>none</systemChrome>
        <transparent>true</transparent>
        <visible>true</visible>
        <minSize>320 240</minSize>
    </initialWindow>
    <installFolder>Example Co/Hello World</installFolder>
    <programMenuFolder>Example Co</programMenuFolder>
    <icon>
        <image16x16>icons/smallIcon.png</image16x16>
        <image32x32>icons/mediumIcon.png</image32x32>
        <image48x48>icons/bigIcon.png</image48x48>
        <image128x128>icons/biggestIcon.png</image128x128>
    </icon>
    <customUpdateUI>true</customUpdateUI>
    <allowBrowserInvocation>false</allowBrowserInvocation>
    <fileTypes>
        <fileType>
            <name>adobe.VideoFile</name>
            <extension>avf</extension>
            <description>Adobe Video File</description>
            <contentType>application/vnd.adobe.video-file</contentType>
            <icon>
                <image16x16>icons/avfIcon_16.png</image16x16>
                <image32x32>icons/avfIcon_32.png</image32x32>
                <image48x48>icons/avfIcon_48.png</image48x48>
                <image128x128>icons/avfIcon_128.png</image128x128>
            </icon>
        </fileType>
    </fileTypes>
</application>
```

## aspectRatio

**Adobe AIR 2.0** 和更新版本， **iPhone** 和 **Android —** 可选

指定应用程序的初始长宽比 .

如果未指定，则应用程序会以设备的 “ 自然 ” 高宽比和方向打开。自然方向因设备而异。通常，在小屏幕设备 （如手机）上是以纵向高宽比打开。在某些设备 （如 iPad 平板电脑）上，应用程序会以当前的方向打开。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

portrait 或 landscape

### 示例

```
<aspectRatio>landscape</aspectRatio>
```

## autoOrients

**Adobe AIR 2.0** 和更新版本， **iPhone** 和 **Android —** 可选

指定应用程序中的内容的方向是否随设备自身更改物理方向而自动重新取向。有关更多信息，请参阅舞台方向。

使用自动定向时，请考虑将舞台的 align 和 scaleMode 属性设置为以下值：

```
stage.align = StageAlign.TOP_LEFT;
stage.scaleMode = StageScaleMode.NO_SCALE;
```

这些设置可以使应用程序沿左上角旋转并防止应用程序内容自动缩放。其他的缩放模式在对内容进行调整以适应旋转的舞台尺寸，同时还会剪辑、扭曲或过度收缩该内容。通常您自己可以通过重绘或转播内容实现更佳的效果。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

true 或 false （默认）

### 示例

```
<autoOrients>true</autoOrients>
```

## colorDepth

**Adobe AIR 3** 和更高版本 **—** 可选

指定是使用 16 位颜色还是 32 位颜色。

使用 16 位颜色可提高渲染性能，但是会牺牲颜色保真度。在 AIR 3 之前， Android 上始终使用 16 位颜色。在 AIR 3 中，默认使用 32 位颜色。

注：如果您的应用程序使用 StageVideo 类，则必须使用 32 位颜色。

父元素：第 157 页的 “android”

子元素：无

内容

以下值之一：

- 16 位
- 32 位

示例

```
<android>
    <colorDepth>16bit</colorDepth>
    <manifestAdditions>...</manifestAdditions>
</android>
```

## content

**Adobe AIR 1.0** 和更新版本 **—** 必需

为 content 元素指定的值是应用程序主内容文件的 URL。该文件可以是 SWF 文件，也可以是 HTML 文件。该 URL 是相对于根应用程序安装文件夹指定的。（如果使用 ADL 运行 AIR 应用程序，则该 URL 相对于包含应用程序描述符文件的文件夹。可以使用 ADL 的 root-dir 参数指定其他根目录。）

父元素：第 171 页的 "initialWindow"

子元素：无

内容

相对于应用程序目录的 URL。因为将 content 元素的值视为 URL，所以必须根据 RFC 1738 中定义的规则对内容文件名称中的字符进行 URL 编码。例如，空格字符必须编码为 %20。

示例

```
<content>TravelPlanner.swf</content>
```

## contentType

**Adobe AIR 1.0** 至 **1.1 —** 可选； **AIR 1.5** 和更新版本 **—** 必需

contentType 是 AIR 1.5 和更新版本所必需的 （在 AIR 1.0 和 1.1 中是可选的）。该属性可以帮助某些操作系统寻找最好的应用程序来打开文件。该值应为文件内容的 MIME 类型。注意，如果文件类型已注册，且具有已分配的 MIME 类型，则在 Linux 中将忽略该值。

父元素：第 166 页的 "fileType"

子元素：无

内容

MIME 类型和子类型。有关 MIME 类型的更多信息，请参阅 RFC2045。

示例

```
<contentType>text/plain</contentType>
```

## copyright

**Adobe AIR 1.0** 和更新版本 **—** 可选

AIR 应用程序的版权信息。在 Mac OS 中，版权文本会显示在已安装应用程序的 " 关于 " 对话框中。在 Mac OS 中，在应用程序的 Info.plist 文件中的 NSHumanReadableCopyright 字段内也使用版权信息。

父元素：第 158 页的 "application"

子元素：无

### 内容

包含应用程序版权信息的字符串。

### 示例

```
<copyright>© 2010, Examples, Inc.All rights reserved.</copyright>
```

## customUpdateUI

**Adobe AIR 1.0** 和更新版本 **—** 可选

指示应用程序是否会提供自己的更新对话框。如果是 false，则 AIR 会向用户提供标准更新对话框。只有作为 AIR 文件发布的应用程序才可以使用内置 AIR 更新系统。

如果应用程序的已安装版本将 customUpdateUI 元素设置为 true，则当用户双击新版本的 AIR 文件或使用无缝安装功能安装应用程序的更新时，运行时将打开应用程序的已安装版本。运行时不会打开默认的 AIR 应用程序安装程序。那么，您的应用程序逻辑可以确定如何继续执行更新操作。（AIR 文件中的应用程序 ID 和发行商 ID 必须与已安装应用程序中的值匹配才能继续进行升级。）

注：仅当应用程序已经安装并且用户双击包含更新的 AIR 安装文件或使用无缝安装功能安装应用程序的更新时，customUpdateUI 机制才能发挥作用。无论 customUpdateUI 是否为 true，您都可以通过您自己的应用程序逻辑下载并启动更新，并在必要时显示自定义 UI。

有关详细信息，请参阅第 197 页的 " 更新 AIR 应用程序 "。

父元素：第 158 页的 "application"

子元素：无

### 内容

true 或 false （默认）

### 示例

```
<customUpdateUI>true</customUpdateUI>
```

## description

**Adobe AIR 1.0** 和更新版本 **—** 可选

显示在 AIR 应用程序安装程序中的应用程序说明。

如果指定单个文本节点 （而非多个文本元素），则无论系统语言为哪种语言， AIR 应用程序安装程序都将使用此说明。否则，AIR 应用程序安装程序会使用与用户操作系统的用户界面语言最匹配的说明。例如，假如在某一安装中，应用程序描述符文件的 description 元素包含适用于 en （英语）区域设置的值。如果用户系统将 en （英语）标识为用户界面语言，则 AIR 应用程序安装程序将使用此 en 说明。如果系统用户界面语言为 en-US （美式英语），则该应用程序也使用此 en 说明。但是，如果系

统用户界面语言为 en-US，而应用程序描述符文件同时定义了 en-US 名称和 en-GB 名称，则 AIR 应用程序安装程序将使用相应的 en-US 值。如果应用程序定义的任何说明与系统用户界面语言均不匹配，则 AIR 应用程序安装程序将使用在应用程序描述符文件中定义的第一个 description 值。

有关开发多语言应用程序的详细信息，请参阅第 228 页的 " 本地化 AIR 应用程序 "。

父元素：第 158 页的 "application"

子元素：第 181 页的 "text"

内容

AIR 1.0 应用程序描述符架构只允许为该名称定义一个简单文本节点 （而非多个 text 元素）。

在 AIR 1.1 （或更高版本）中，您可以在 description 元素中指定多种语言。每个文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

示例

使用简单文本节点的说明：

```
<description>This is a sample AIR application.</description>
```

使用针对英语、法语和西班牙语的本地化文本元素的说明 （AIR 1.1 和更新版本中有效）：

```
<description>
    <text xml:lang="en">This is an example.</text>
    <text xml:lang="fr">C'est un exemple.</text>
    <text xml:lang="es">Esto es un ejemplo.</text>
</description>
```

## description

**Adobe AIR 1.0** 和更新版本 **—** 必需

操作系统向用户显示文件类型说明。文件类型说明不可本地化。

另请参阅 第 163 页的 "description"，它是 application 元素的子元素

父元素：第 166 页的 "fileType"

子元素：无

内容

描述文件内容的字符串。

示例

```
<description>PNG image</description>
```

## extension

**Adobe AIR 1.0** 和更新版本 **—** 必需

文件类型的扩展字符串。

父元素：第 166 页的 "fileType"

子元素：无

内容

标识文件扩展字符的字符串 （没有 “.”）

示例

```
<extension>png</extension>
```

## extensionID

仅限 **Adobe AIR 2.5** 和更高版本、 **tv** 以及 **extendedTV** 配置文件 **—** 必需

指定应用程序所使用的 ActionScript 扩展的 ID。该 ID 是在扩展名描述符文档中定义的。

父元素：第 165 页的 “extensions”

子元素：无

内容

标识 ActionScript 扩展 ID 的字符串。

示例

```
<extensionID>com.example.extendedFeature</extensionID>
```

## extensions

仅限 **Adobe AIR 2.5** 和更高版本、 **tv** 以及 **extendedTV** 配置文件 **—** 可选

标识应用程序所使用的 ActionScript 扩展。

父元素：第 158 页的 “application”

子元素：第 165 页的 “extensionID”

内容

extensionID 子元素，其中包含扩展描述符文件中的 ActionScript 扩展 ID。

示例

```
<extensions>
    <extensionID>extension.first</extensionID>
    <extensionID>extension.next</extensionID>
    <extensionID>extension.last</extensionID>
</extensions>
```

## filename

**Adobe AIR 1.0** 和更新版本 **—** 必需

该字符串在安装应用程序时用作应用程序的文件名 （不带扩展名）。该应用程序文件将在运行时中启动 AIR 应用程序。如果未提供 name 值，则 filename 也将用作安装文件夹的名称。

父元素：第 158 页的 “application”

子元素：无

内容

filename 属性可包含任何 Unicode (UTF-8) 字符，但以下字符 （在各种文件系统中都禁止将这些字符用作文件名）除外：

| 字符 | 十六进制代码 |
|---|---|
| 因系统而异 | 0x00 - x1F |
| * | x2A |
| " | x22 |
| : | x3A |
| > | x3C |
| < | x3E |
| ? | x3F |
| \ | x5C |
| \| | x7C |

filename 值不能以句点结尾。

示例

```
<filename>MyApplication</filename>
```

## fileType

**Adobe AIR 1.0** 和更新版本 **—** 可选

描述应用程序可以注册的单一文件类型。

父元素：第 167 页的 "fileTypes"

子元素：

- 第 162 页的 "contentType"
- 第 164 页的 "description"
- 第 164 页的 "extension"
- 第 168 页的 "icon"
- 第 177 页的 "name"

内容

描述文件类型的元素。

示例

```
<fileType>
    <name>foo.example</name>
    <extension>foo</extension>
    <description>Example file type</description>
    <contentType>text/plain</contentType>
    <icon>
        <image16x16>icons/fooIcon16.png</image16x16>
        <image48x48>icons/fooIcon48.png</imge48x48>
    <icon>
</fileType>
```

## fileTypes

**Adobe AIR 1.0** 和更新版本 **—** 可选

利用 fileTypes 元素可声明 AIR 应用程序可以与其关联的文件类型。

当安装某个 AIR 应用程序时，会在操作系统中注册任何已声明的文件类型。如果这些文件类型尚未与其他应用程序关联，则它们将与该 AIR 应用程序关联。若要覆盖某个文件类型和其他应用程序之间的现有关联，请在运行时使用 NativeApplication.setAsDefaultApplication() 方法 （最好使用用户权限）。

注：运行时方法只能管理应用程序描述符中声明的文件类型的关联。

fileTypes 元素为可选元素。

父元素：第 158 页的 “application”

子元素：第 166 页的 “fileType”

内容

fileTypes 元素可以包含任何数量的 fileType 元素。

示例

```
<fileTypes>
    <fileType>
        <name>adobe.VideoFile</name>
        <extension>avf</extension>
        <description>Adobe Video File</description>
        <contentType>application/vnd.adobe.video-file</contentType>
        <icon>
            <image16x16>icons/AIRApp_16.png</image16x16>
            <image32x32>icons/AIRApp_32.png</image32x32>
            <image48x48>icons/AIRApp_48.png</image48x48>
            <image128x128>icons/AIRApp_128.png</image128x128>
        </icon>
    </fileType>
</fileTypes>
```

## fullScreen

**Adobe AIR 2.0** 和更新版本， **iPhone** 和 **Android —** 可选

指定是否用全屏模式启动应用程序。

父元素：第 171 页的 “initialWindow”

子元素：无

内容
true 或 false （默认）

示例
```
<fullscreen>true</fullscreen>
```

## height

**Adobe AIR 1.0** 和更新版本 **—** 可选

应用程序的主窗口的初始高度。

如果您没有设置高度，则高度会由根 SWF 文件中的设置来确定，如果是基于 HTML 的 AIR 应用程序，则由操作系统来确定。

在 AIR 2 中，窗口的最大高度范围是 2048 像素至 4096 像素。

父元素：第 171 页的 “initialWindow”

子元素：无

内容
最大值为 4095 的正无穷整数。

示例
```
<height>4095</height>
```

## icon

**Adobe AIR 1.0** 和更新版本 **—** 可选

icon 属性指定一个或多个要用于应用程序的图标文件。包含图标是可选的。如果未指定 icon 属性，则操作系统将显示默认图标。

指定路径相对于应用程序的根目录。图标文件必须为 PNG 格式。可以指定以下所有图标尺寸：

如果存在用于指定一定尺寸的元素，则文件中的图像必须与指定尺寸完全相同。如果所有尺寸都未提供，则操作系统会将图像缩放为适合图标给定用途的最接近的大小。

注：指定图标不会自动添加到 AIR 包中。打包应用程序时，图标文件必须包含在其正确的相对位置中。

为获得最佳效果，为每种可用尺寸都提供一个图像。此外，请确保图标在 16 位和 32 位颜色模式下看上去都像一回事。

父元素：第 158 页的 “application”

子元素：第 169 页的 “imageNxN”

内容
针对每个所需的图标大小的 imageNxN 元素。

示例

```
<icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.png</image32x32>
    <image48x48>icons/bigIcon.png</image48x48>
    <image128x128>icons/biggestIcon.png</image128x128>
</icon>
```

# id

**Adobe AIR 1.0** 和更新版本 **—** 必需

应用程序的标识符字符串，称为应用程序 ID。通常会使用相反的 DNS 样式标识符，但该样式不是必需的。

父元素：第 158 页的 “application”

子元素：无

内容

该 ID 值只能使用以下字符：

- 0 - 9
- a - z
- A - Z
- .（点）
- -（连字符）

该值必须包含 1 到 212 个字符。此元素是必需的。

示例

```
<id>org.example.application</id>
```

# imageNxN

**Adobe AIR 1.0** 和更新版本 **—** 可选

定义指向应用程序目录对应图标的路径。

可以使用以下图标图像，每个图像指定一个不同的图标大小：

- image16x16
- image29x29 (AIR 2+)
- image32x32
- image36x36 (AIR 2.5+)
- image48x48
- image57x57 (AIR 2+)
- image72x72 (AIR 2+)
- image114x114 (AIR 2.6+)
- image128x128
- image512x512 (AIR 2+)

图标必须是图像元素指示的确切大小的 PNG 图形。图标文件必须包括在应用程序包中；应用程序描述符文档引用的图标不会自动包括进去。

父元素：第 158 页的 “application”

子元素：无

内容
指向图标的文件路径可以包含任何 Unicode (UTF-8) 字符，以下禁止用作各种文件系统中的文件名的字符除外：

| 字符 | 十六进制代码 |
| --- | --- |
| 因系统而异 | 0x00 - x1F |
| * | x2A |
| " | x22 |
| : | x3A |
| > | x3C |
| < | x3E |
| ? | x3F |
| \ | x5C |
| \| | x7C |

示例
```
<image32x32>icons/icon32.png</image32x32>
```

## InfoAdditions

**Adobe AIR 1.0** 和更新版本 **—** 可选

可以使您指定 iPhone 应用程序的其他属性。

父元素：第 173 页的 “iPhone”

子元素：iPhone Info.plist elements

内容
包含的子元素可指定用作应用程序的 Info.plist 设置的键值对。 InfoAdditions 元素的内容应该包括在 CDATA 区块中。

有关键值对以及在 XML 中如何表达的信息，请参阅 Apple iPhone 参考库中的信息属性列表键参考。

示例
```
<InfoAdditions>
    <![CDATA[
        <key>UIStatusBarStyle</key>
        <string>UIStatusBarStyleBlackOpaque</string>
        <key>UIRequiresPersistentWiFi</key>
        <string>NO</string>
    ]]>
</InfoAdditions>
```

**更多帮助主题**

第 69 页的 "iOS 设置 "

## initialWindow

**Adobe AIR 1.0** 和更新版本 — 必需

定义主要内容文件和初始应用程序外观。

父元素：第 158 页的 "application"

子元素：以下所有元素可以显示为 initialWindow 元素的子元素。但是，根据 AIR 是否在平台上支持 windows，某些元素会被忽略。

| 元素 | 桌面 | 优化 Adobe® Flash® Platform 的性能 | TV |
|---|---|---|---|
| 第 161 页的 "aspectRatio" | 忽略 | 使用 | 忽略 |
| 第 161 页的 "autoOrients" | 忽略 | 使用 | 忽略 |
| 第 162 页的 "content" | 使用 | 使用 | 使用 |
| 第 167 页的 "fullScreen" | 忽略 | 使用 | 使用 |
| 第 168 页的 "height" | 使用 | 忽略 | 忽略 |
| 第 175 页的 "maximizable" | 使用 | 忽略 | 忽略 |
| 第 175 页的 "maxSize" | 使用 | 忽略 | 忽略 |
| 第 176 页的 "minimizable" | 使用 | 忽略 | 忽略 |
| 第 176 页的 "minSize" | 使用 | 忽略 | 忽略 |
| 第 178 页的 "renderMode" | 忽略 | 使用 | 忽略 |
| 第 179 页的 "resizable" | 使用 | 忽略 | 忽略 |
| 第 179 页的 "softKeyboardBehavior" | 忽略 | 使用 | 忽略 |
| 第 181 页的 "systemChrome" | 使用 | 忽略 | 忽略 |
| 第 182 页的 "title" | 使用 | 忽略 | 忽略 |

| 元素 | 桌面 | 优化 Adobe® Flash® Platform 的性能 | TV |
|---|---|---|---|
| 第 182 页的 “transparent” | 使用 | 忽略 | 忽略 |
| 第 183 页的 “visible” | 使用 | 忽略 | 忽略 |
| 第 184 页的 “width” | 使用 | 忽略 | 忽略 |
| 第 184 页的 “x” | 使用 | 忽略 | 忽略 |
| 第 185 页的 “y” | 使用 | 忽略 | 忽略 |

内容

定义应用程序外观和行为的子元素。

示例

```
<initialWindow>
    <title>Hello World</title>
    <content>
        HelloWorld.swf
    </content>
    <systemChrome>none</systemChrome>
    <transparent>true</transparent>
    <visible>true</visible>
    <maxSize>1024 800</maxSize>
    <minSize>320 240</minSize>
    <maximizable>false</maximizable>
    <minimizable>false</minimizable>
    <resizable>true</resizable>
    <x>20</x>
    <y>20</y>
    <height>600</height>
    <width>800</width>
    <aspectRatio>landscape</aspectRatio>
    <autoOrients>true</autoOrients>
    <fullScreen>false</fullScreen>
    <renderMode>auto</renderMode>
</initialWindow>
```

## installFolder

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定默认安装目录的子目录。

在 Windows 中，默认安装子目录为 Program Files 目录。在 Mac OS 中，默认安装子目录为 /Applications 目录。在 Linux 中为 /opt/。例如，如果将 installFolder 属性设置为 "Acme"，并将应用程序命名为 "ExampleApp"，则应用程序在 Windows 中将安装在 C:\Program Files\Acme\ExampleApp 中，在 MacOS 中将安装在 /Applications/Acme/Example.app 中，而在 Linux 中将安装在 /opt/Acme/ExampleApp 中。

installFolder 属性为可选属性。如果未指定 installFolder 属性，则应用程序将根据 name 属性安装在默认安装目录的子目录中。

父元素：第 158 页的 “application”

子元素：无

内容

installFolder 属性可以包含任何 Unicode (UTF-8) 字符，但那些禁止在各种文件系统用作文件夹名称的字符除外 （有关例外字符的列表，请参阅 filename 属性）。

如果要指定嵌套子目录，请使用正斜杠 (/) 字符作为目录分隔符。

示例

```
<installFolder>utilities/toolA</installFolder>
```

## iPhone

**Adobe AIR 2.0**，只针对 **iPhone —** 可选

指定 iPhone 特定应用程序属性。

父元素：第 158 页的 “application”

子元素：

- 第 170 页的 “InfoAdditions”
- 第 179 页的 “requestedDisplayResolution”

### 更多帮助主题

第 69 页的 “iOS 设置 ”

## manifest

**Adobe AIR 2.5** 和更新版本，只针对 **Android —** 可选

为应用程序指定添加到 Android 清单文件的信息。

父元素：第 174 页的 “manifestAdditions”

子元素：由 Android SDK 定义。

内容

从技术方面而言， manifest 元素并不是 AIR 应用程序描述符方案的一部分。它是 Android 清单 XML 文档的根。放入 manifest 元素中的任何内容都必须符合 AndroidManifest.xml 方案。当使用 AIR 工具生成 APK 文件时，会将 manifest 元素中的信息复制到所生成的应用程序 AndroidManifest.xml 的相应部分。

manifest 元素自身必须包括在 AIR 应用程序描述符内的 CDATA 块中。

示例

```
<![CDATA[
    <manifest android:sharedUserID="1001">
        <uses-permission android:name="android.permission.CAMERA"/>
        <uses-feature android:required="false" android:name="android.hardware.camera"/>
        <application android:allowClearUserData="true"
                    android:enabled="true"
                    android:persistent="true"/>
    </manifest>
]]>
```

**更多帮助主题**

第 64 页的 “Android 设置 ”

AndroidManifest.xml 文件

## manifestAdditions

**Adobe AIR 2.5** 和更新版本，只针对 **Android**

指定要添加到 Android 清单文件的信息。

每个 Android 应用程序都包括一个定义基本应用程序属性的清单文件。 Android 清单在概念上和 AIR 应用程序描述符相似。AIR for Android 应用程序既具有应用程序描述符也具有自动生成的 Android 清单文件。在对 AIR for Android 应用程序进行打包时，会将此 manifestAdditions 元素中的信息添加到 Android 清单文档的相应部分。

父元素：第 157 页的 “android”

子元素：第 173 页的 “manifest”

内容

manifestAdditions 元素中的信息会添加到 AndroidManifest XML 文档中。

AIR 会在生成的 Android 清单文档中设置多个清单项目，以确保应用程序和运行时功能正确运行。不能覆盖以下设置：

不能设置 manifest 元素的以下属性：

- package
- android:versionCode
- android:versionName

不能设置主 activity 元素的以下属性：

- android:label
- android:icon

不能设置 application 元素的以下属性：

- android:theme
- android:name
- android:label
- android:windowSoftInputMode
- android:configChanges
- android:screenOrientation
- android:launchMode

示例

```
<manifestAdditions>
    <![CDATA[
        <manifest android:installLocation="preferExternal">
            <uses-permission android:name="android.permission.INTERNET"/>
            <application android:allowClearUserData="true"
                        android:enabled="true"
                        android:persistent="true"/>
        </manifest>
    ]]>
</manifestAdditions>
```

**更多帮助主题**

第 64 页的 “Android 设置 ”

AndroidManifest.xml 文件

# maximizable

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定窗口是否可最大化。

注：在操作系统 （例如 Mac OS X）中，最大化窗口是一种调整大小操作，若要阻止窗口缩放或调整大小， maximizable 和 resizable 必须同时设置为 false。

父元素：第 171 页的 “initialWindow”

子元素：无

内容

true （默认）或 false

示例

```
<maximizable>false</maximizable>
```

# maxSize

**Adobe AIR 1.0** 和更新版本 **—** 可选

窗口的最大尺寸。如果您没有设置最大尺寸，将由操作系统来确定。

父元素：第 171 页的 “initialWindow”

子元素：无

内容

两个整数分别代表最大宽度和最大高度，以空格分隔。

注：AIR 支持的最大窗口尺寸在 AIR 2 中自 2048x2048 像素增加到 4096x4096 像素。(因为屏幕坐标是基于零的，所以您能使用的最大宽度或最大高度值是 4095）。

示例

```
<maxSize>1024 360</maxSize>
```

## minimizable

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定窗口是否可最小化。

父元素：第 171 页的 “initialWindow”

子元素：无

内容

true （默认）或 false

示例

```
<minimizable>false</minimizable>
```

## minSize

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定允许的最小窗口尺寸。

父元素：第 171 页的 “initialWindow”

子元素：无

内容

两个整数分别代表最小宽度和最小高度，以空格分隔。请注意操作系统强制的最小尺寸会优先于应用程序描述符中设置的值。

示例

```
<minSize>120 60</minSize>
```

## name

**Adobe AIR 1.0** 和更新版本 **—** 可选

AIR 应用程序安装程序显示的应用程序标题。

如果未指定 name 元素，则 AIR 应用程序安装程序会将 filename 显示为应用程序名称。

父元素：第 158 页的 “application”

子元素：第 181 页的 “text”

内容

如果指定单个文本节点 （而非多个 <text> 元素），则无论系统语言为哪种语言， AIR 应用程序安装程序都将使用此名称。

AIR 1.0 应用程序描述符架构只允许为该名称定义一个简单文本节点 （而非多个 text 元素）。在 AIR 1.1 （或更高版本）中，您可以在 name 元素中指定多种语言。

每个文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

AIR 应用程序安装程序会使用与用户操作系统的用户界面语言最匹配的名称。例如，假如在某一安装中，应用程序描述符文件的 name 元素包含适用于 en （英语）区域设置的值。如果操作系统将 en （英语）标识为用户界面语言，则 AIR 应用程序安装程序将使用此 en 名称。如果系统用户界面语言为 en-US （美式英语），则该应用程序也使用此 en 名称。但是，如果用户界面

语言为 en-US，而应用程序描述符文件同时定义了 en-US 名称和 en-GB 名称，则 AIR 应用程序安装程序将使用相应的 en-US 值。如果应用程序定义的任何名称与系统用户界面语言均不匹配，则 AIR 应用程序安装程序将使用在应用程序描述符文件中定义的第一个 name 值。

name 元素仅定义在 AIR 应用程序安装程序中使用的应用程序标题。 AIR 应用程序安装程序支持以下多种语言：繁体中文、简体中文、捷克语、荷兰语、英语、法语、德语、意大利语、日语、朝鲜语、波兰语、巴西葡萄牙语、俄语、西班牙语、瑞典语和土耳其语。 AIR 应用程序安装程序将根据系统用户界面语言来（为文本而不是应用程序标题和说明）选择其显示语言。此语言选择与应用程序描述符文件中的设置无关。

name 元素不定义可供运行的已安装应用程序使用的区域设置。有关开发多语言应用程序的详细信息，请参阅第 228 页的 " 本地化 AIR 应用程序 "。

示例

以下示例使用简单文本节点定义名称。

```
<name>Test Application</name>
```

AIR 1.1 和更新版本中有效的以下示例使用 <text> 元素节点指定三种语言的 （英语、法语和西班牙语）名称：

```
<name>
    <text xml:lang="en">Hello AIR</text>
    <text xml:lang="fr">Bonjour AIR</text>
    <text xml:lang="es">Hola AIR</text>
</name>
```

## name

**Adobe AIR 1.0** 和更新版本 **—** 必需

指定文件类型的名称。

父元素：第 166 页的 "fileType"

子元素：无

内容

表示文件类型名称的字符串。

示例

```
<name>adobe.VideoFile</name>
```

## programMenuFolder

**Adobe AIR 1.0** 和更新版本 **—** 可选

（可选）标识应用程序的快捷方式在 Windows 操作系统的 " 所有程序 " 菜单中或 Linux 的 " 应用程序 " 菜单中放置的位置。（目前在其他操作系统中忽略此设置。）

父元素：第 158 页的 "application"

子元素：无

内容

用于 programMenuFolder 值的字符串可以包含任何 Unicode (UTF-8) 字符，但那些禁止在各种文件系统用作文件夹名称的字符除外 （有关例外字符的列表，请参阅 filename 元素）。请勿 将正斜杠 (/) 字符用作此值的最后一个字符。

示例

```
<programMenuFolder>Example Company/Sample Application</programMenuFolder>
```

## publisherID

**Adobe AIR 1.5.3** 和更新版本 **—** 可选

标识最初由 AIR 1.5.2 或更早版本创建的用于更新 AIR 应用程序的发行商 ID。

创建应用程序更新时只能指定一个发行商 ID。 publisherID 元素的值必须与 AIR 生成的针对更早版本的应用程序的发行商 ID 相匹配。针对已安装的应用程序，发行商 ID 可以在 META-INF/AIR/publisherid 文件中的安装应用程序的文件夹中找到。

使用 AIR 1.5.3 或更新版本创建的新应用程序不需要指定发行商 ID。

有关详细信息，请参阅第 144 页的 " 关于 AIR 发行商标识符 "。

父元素：第 158 页的 "application"

子元素：无

内容

发行商 ID 字符串。

示例

```
<publisherID>B146A943FBD637B68C334022D304CEA226D129B4.1</publisherID>
```

## renderMode

**Adobe AIR 2.0** 和更新版本 **—** 可选

如果目前的计算设备支持，指定是否使用图形处理单元 (GPU) 加速。

父元素：第 171 页的 "initialWindow"

子元素：无

内容

以下值之一：

- auto （默认）— 目前回退到 CPU 模式。
- cpu — 未使用硬件加速。
- direct — CPU 中进行的渲染合成；使用 GPU 进行块传输。在 AIR 3+ 中可用。
- gpu — 如果可用的话，使用硬件加速。

  重要说明：请勿对 Flex 应用程序使用 GPU 渲染模式。

示例

```
<renderMode>direct</renderMode>
```

## requestedDisplayResolution

**Adobe AIR 2.6** 和更高版本，只针对 **iOS —** 可选

指定在具有高分辨率屏幕的 iOS 设备上，应用程序是希望使用标准分辨率还是希望使用高分辨率。当设置为 standard （默认值）时，该屏幕将作为标准分辨率屏幕显示给应用程序。即，全屏舞台尺寸为 320x480，即便是在 640x960 高分辨率屏幕上也是如此。当设置为 high 时，应用程序可以处理每一个高分辨率像素。即，全屏舞台尺寸为 640x960。

在具有标准分辨率屏幕的设备上，无论使用哪一种设置，舞台尺寸都将与屏幕尺寸相匹配。

父元素：第 173 页的 “iPhone”

子元素：无

### 内容

standard （默认值）或 high。

### 示例

```
<requestedDisplayResolution>high</requestedDisplayResolution>
```

## resizable

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定窗口是否可调整大小。

注：在操作系统 （例如 Mac OS X）中，最大化窗口是一种调整大小操作，若要阻止窗口缩放或调整大小， maximizable 和 resizable 必须同时设置为 false。

父元素：第 171 页的 “initialWindow”

子元素：

### 内容

true （默认）或 false

### 示例

```
<resizable>false</resizable>
```

## softKeyboardBehavior

**Adobe AIR 2.6** 和更高版本，移动配置文件 **—** 可选

指定当显示虚拟键盘时应用程序的默认行为。默认行为是向上平移应用程序。运行时在屏幕上保持具有焦点的文本字段或交互式对象。如果您的应用程序未提供自己的键盘处理逻辑，请使用 pan 选项。

通过将 softKeyboardBehavior 元素设置为 none，还可以关闭自动行为。在这种情况下，当软键盘浮现时，文本字段和交互式对象将调度 SoftKeyboardEvent，但运行时不会平移应用程序或调整应用程序大小。您的应用程序负责将文本输入区域保持在可见区域内。

父元素：第 158 页的 “application”

子元素：无

内容

none 或 pan。默认值为 pan。

示例

```
<softKeyboardBehavior>none</softKeyboardBehavior>
```

**更多帮助主题**

SoftKeyboardEvent

## supportedProfiles

**Adobe AIR 2.0** 和更新版本 — 可选

标识支持应用程序的配置文件。

父元素：第 158 页的 “application”

子元素：无

内容

supportedProfiles 元素可以包括以下任何值：

- desktop — desktop 是安装在使用 AIR 文件的 desktop 计算机上的 AIR 应用程序的配置文件。这些应用程序无权访问 NativeProcess 类（提供与本机应用程序的通信）。
- extendedDesktop — 扩展的桌面配置文件是安装在使用本机应用程序安装程序的桌面计算机上的 AIR 应用程序的配置文件。这些应用程序可以访问 NativeProcess 类（提供与本机应用程序的通信）。
- mobileDevice — 移动设备配置文件用于移动应用程序。
- extendedMobileDevice — 扩展的移动设备配置文件当前未被使用。
- tv — tv 配置文件用于使用 AIR 文件安装在电视设备上的应用程序。
- extendedTV — extendedTV 配置文件用于使用 AIRN 文件安装在电视设备上的应用程序。这些应用程序可以访问 ActionScript 扩展。

supportedProfiles 属性是可选的。当应用程序描述符文件中不包括此元素时，应用程序可以针对任何配置文件进行编译和部署。

要指定多个配置文件，请使用空格字符将配置文件彼此隔开。例如，以下设置指定此应用程序仅适用于桌面配置文件和扩展配置文件：

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

注：当您运行带有 ADL 的应用程序，并且未指定 ADL -profile 选项的值时，系统会使用应用程序描述符中的第一个配置文件。（如果应用程序描述符中也没有指定配置文件，系统会使用桌面配置文件）。

示例

```
<supportedProfiles>desktop mobileDevice</supportedProfiles>
```

**更多帮助主题**

第 186 页的 “ 设备配置文件 ”

第 63 页的 “ 支持的配置文件 ”

## systemChrome

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定是否使用操作系统提供的标准标题栏、边框和控件创建初始应用程序窗口。

系统窗口样式设置在运行时无法更改。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

以下值之一：

- none — 未提供系统窗口样式。应用程序 （或者如 Flex 之类的应用程序框架）负责显示窗口样式。
- standard （默认）— 系统窗口样式由操作系统提供。

### 示例

```
<systemChrome>standard</systemChrome>
```

## text

**Adobe AIR 1.1** 和更新版本 **—** 可选

指定本地化字符串。

文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

AIR 应用程序安装程序使用具有与用户操作系统的用户界面语言最匹配的 xml:lang 属性值的 text 元素。

例如，考虑一种安装， text 元素将针对 en （英语）区域设置的值包括在该安装中。如果操作系统将 en （英语）标识为用户界面语言，则 AIR 应用程序安装程序将使用此 en 名称。如果系统用户界面语言为 en-US （美式英语），则该应用程序也使用此 en 名称。但是，如果用户界面语言为 en-US，而应用程序描述符文件同时定义了 en-US 名称和 en-GB 名称，则 AIR 应用程序安装程序将使用相应的 en-US 值。

如果应用程序定义的任何 text 元素与系统用户界面语言均不匹配，则 AIR 应用程序安装程序将使用在应用程序描述符文件中定义的第一个 name 值。

父元素：

- 第 176 页的 “name”
- 第 163 页的 “description”

子元素：无

### 内容

指定区域设置和本地化文本字符串的 xml:lang 属性。

### 示例

```
<text xml:lang="fr">Bonjour AIR</text>
```

## title

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定显示在初始应用程序窗口的标题栏内的标题。

如果 systemChrome 元素设置为 standard，那么只显示一个标题。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

包含窗口标题的字符串。

### 示例

```
<title>Example Window Title</title>
```

## transparent

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定初始应用程序窗口是否与桌面进行 alpha 混合。

透明窗口绘制起来可能比较慢且需要更多内存。透明设置在运行时无法更改。

重要说明：当 systemChrome 为 none 时，只能将 transparent 设置为 true。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

true 或 false （默认）

### 示例

```
<transparent>true</transparent>
```

## version

**Adobe AIR 1.0** 至 **2.0 —** 必需； **AIR 2.5** 和更新版本中不允许

指定应用程序的版本信息。

version 字符串是应用程序定义的指示符。 AIR 不会以任何方式解释版本字符串。因此，不会假设版本 “3.0” 比版本 “2.0” 更新。示例包括：“1.0”、 “.4”、 “0.5”、 “4.9” 和 “1.3.4a”。

在 AIR 2.5 和更新版本中， versionNumber 和 versionLabel 元素取代了 version 元素。

父元素：第 158 页的 “application”

子元素：无

### 内容

包含应用程序版本的字符串。

示例

```
<version>0.1 Alpha</version>
```

## versionLabel

**Adobe AIR 2.5** 和更新版本 **—** 可选

指定一种人类可读的版本字符串。

取代 versionNumber 元素值在安装对话框中显示的是版本标签值。如果未使用 versionLabel，那么两者都会使用 versionNumber。

父元素：第 158 页的 “application”

子元素：无

内容

包含公开显示的版本文本的字符串。

示例

```
<versionLabel>0.9 Beta</versionlabel>
```

## versionNumber

**Adobe AIR 2.5** 和更新版本 **—** 必需

应用程序版本号。

父元素：第 158 页的 “application”

子元素：无

内容

版本号可以包含按时期分隔的三个整数组成的序列。每个整数必须是 0 到 999 （两者都包括）间的数字。

示例

```
<versionNumber>1.0.657</versionNumber>

<versionNumber>10</versionNumber>

<versionNumber>0.01</versionNumber>
```

## visible

**Adobe AIR 1.0** 和更新版本 **—** 可选

指定初始应用程序窗口是否在创建后即可见。

AIR 窗口，包括初始窗口，默认情况下是以不可见的状态创建的。可以通过对此本机窗口对象调用 activate() 方法或将 visible 属性设置为 true 来显示窗口。您可能想要使主窗口最初保持隐藏，以便不显示对窗口位置、窗口大小和其内容的布局所做的更改。

除非在 MXML 定义中将 visible 属性设置为 false，否则 Flex mx:WindowedApplication 组件会自动显示并激活窗口，然后立即调度 applicationComplete 事件。

在移动和 tv 配置文件 （两者都不支持窗口）中的设备上，可见设置可以忽略。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

true 或 false （默认）

### 示例

```
<visible>true</visible>
```

## width

**Adobe AIR 1.0** 和更新版本 **—** 可选

应用程序的主窗口的初始宽度。

如果您没有设置宽度，则宽度会由根 SWF 文件中的设置来确定，如果是基于 HTML 的 AIR 应用程序，则由操作系统来确定。

在 AIR 2 中，窗口的最大宽度范围是 2048 像素至 4096 像素。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

最大值为 4095 的正无穷整数。

### 示例

```
<width>1024</width>
```

## x

**Adobe AIR 1.0** 和更新版本 **—** 可选

初始应用程序窗口的水平位置。

在大多数情况下，让操作系统来决定窗口的初始位置比分配一个固定值更好。

屏幕坐标系统的原点 (0,0) 是顶部，主桌面屏幕的左角 （如操作系统决定的那样）。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

整型值。

### 示例

```
<x>120</x>
```

## y

**Adobe AIR 1.0** 和更新版本 **—** 可选

初始应用程序窗口的垂直位置。

在大多数情况下，让操作系统来决定窗口的初始位置比分配一个固定值更好。

屏幕坐标系统的原点 (0,0) 是顶部，主桌面屏幕的左角 （如操作系统决定的那样）。

父元素：第 171 页的 “initialWindow”

子元素：无

### 内容

整型值。

### 示例

```
<y>250</y>
```

# 第 15 章：设备配置文件

**Adobe AIR 2** 和更高版本

配置文件是一种机制，用于定义应用程序运行所在的计算设备的类。一个配置文件定义一组 API 和功能，它们通常在特定类的设备上受支持。可用的配置文件包括：

- 桌面
- extendedDesktop
- mobileDevice
- extendedMobileDevice
- tv
- extendedTV

您可以在应用程序描述符中定义应用程序的配置文件。所包含配置文件中的计算机和设备的用户可以安装应用程序，其他计算机和设备的用户则不能。例如，如果在应用程序描述符中仅包含桌面配置文件，则用户只能在桌面计算机上安装和运行应用程序。

如果应用程序并不真正支持所包含的配置文件，则此类环境下的用户体验可能很差。如果在应用程序描述符中没有指定任何配置文件，则 AIR 不会对应用程序施加任何限制。您可以使用任意一种支持的格式来打包应用程序，而且任何配置文件中的设备的用户都可以安装该应用程序，但是它可能无法在运行时正常工作。

系统会尽可能地在您打包应用程序时施加配置文件限制。例如，如果仅包含 extendedDesktop 配置文件，则无法将应用程序打包成 AIR 文件，而只能打包成本机安装程序。同样地，如果仅包含 mobileDevice 配置文件，则无法将应用程序打包成 Android APK。

单个计算设备可支持多个配置文件。例如，桌面计算机上的 AIR 支持桌面配置文件应用程序和 extendedDesktop 配置文件应用程序。但是，扩展的桌面配置文件应用程序可以与本机进程通信，而且必须打包成本机安装程序（exe、dmg、deb 或 rpm）。而桌面配置文件应用程序则无法与本机进程通信。桌面配置文件应用程序可以打包成 AIR 文件或本机安装程序。

在配置文件中包含功能，表示定义了该配置文件的设备类通常支持该功能。但是，这并不表示配置文件中的每种设备支持每种功能。例如，大多数（但并非全部）移动电话都包含一个加速度计。不具有通用支持的类和功能通常有一个布尔属性，您可以在使用功能之前检查该属性。例如，对于加速度计的情况，您可以测试静态属性 Accelerometer.isSupported 以确定当前设备是否拥有支持的加速度计。

在应用程序描述符中使用 supportedProfiles 元素可以将以下配置文件分配给 AIR 应用程序：

**桌面** 桌面配置文件为作为 AIR 文件安装在桌面计算机上的 AIR 应用程序定义一组功能。这些应用程序将在支持的桌面平台（Mac OS、Windows 和 Linux）上安装并运行。在 AIR 2 以前的 AIR 版本中开发的 AIR 应用程序可视为位于桌面配置文件中。某些 API 在此配置文件中无法正常运行。例如，桌面应用程序无法与本机进程通信。

**扩展的桌面** 扩展的桌面配置文件为打包到本机安装程序并随之一起安装的 AIR 应用程序定义一组功能。这些本机安装程序是 Windows 上的 EXE 文件、Mac OS 上的 DMG 文件和 Linux 上的 BIN、DEB 或 RPM 文件。扩展的桌面应用程序具有在桌面配置文件应用程序中不可用的其他功能。有关更多信息，请参阅第 49 页的“对桌面本机安装程序进行打包”。

**移动设备** 移动设备配置文件可为安装在移动设备（例如手机和平板电脑）上的应用程序定义一组功能。这些应用程序可在受支持的移动平台（包括 Android、Blackberry Tablet OS 和 iOS）上安装并运行。

**扩展的移动设备** 扩展的移动设备配置文件为安装在移动设备上的应用程序定义一组扩展功能。目前，没有支持此配置文件的设备。

**TV** 电视配置文件为电视定义一组功能。该配置文件还包含将电视用作主显示器的设备，如蓝光光盘播放机、数码录像机和机顶盒。该配置文件中的应用程序不能使用 Adobe AIR 的本机扩展。

**扩展的电视** 扩展的电视配置文件所涵盖的设备类型与电视配置文件相同，但包括了附加功能，如 Adobe AIR 的本机扩展。

# 限制应用程序描述符文件中的目标配置文件

**Adobe AIR 2** 和更高版本

从 AIR 2 开始，应用程序描述符文件包含一个 supportedProfiles 元素，使用该元素可以限制目标配置文件。例如，以下设置指定此应用程序仅适用于桌面配置文件：

```
<supportedProfiles>desktop</supportedProfiles>
```

设置此元素后，只能在您列出的配置文件中打包应用程序。使用下列值：

- desktop — 桌面配置文件
- extendedDesktop — 扩展的桌面配置文件
- mobileDevice — 移动设备配置文件
- tv - 电视配置文件
- extendedTV - 扩展的电视配置文件

supportedProfiles 元素是可选的。如果应用程序描述符文件中不包括此元素，应用程序可以针对任何配置文件进行打包和部署。

要在 supportedProfiles 元素中指定多个配置文件，请使用空格字符将配置文件彼此隔开，如下所示：

```
<supportedProfiles>desktop extendedDesktop</supportedProfiles>
```

# 不同配置文件的功能

**Adobe AIR 2** 和更高版本

下表列出了所有配置文件中都不支持的类和功能。

| 类或功能 | 桌面 | extendedDesktop | mobileDevice | tv | extendedTV |
|---|---|---|---|---|---|
| Accelerometer (Accelerometer.isSupported) | 否 | 否 | 检查 | 否 | 否 |
| Accessibility (Capabilities.hasAccessibility) | 是 | 是 | 否 | 否 | 否 |
| 回音消除 (Microphone.getEnhancedMicrophone()) | 是 | 是 | 否 | 否 | 否 |
| ActionScript 2 | 是 | 是 | 否 | 是 | 是 |
| CacheAsBitmap 矩阵 | 否 | 否 | 是 | 是 | 是 |
| Camera (Camera.isSupported) | 是 | 是 | 是 | 否 | 否 |
| CameraRoll | 否 | 否 | 是 | 否 | 否 |
| CameraUI (CameraUI.isSupported) | 否 | 否 | 是 | 否 | 否 |
| 捕获运行时捆绑 | 是 | 是 | 是 | 否 | 否 |
| ContextMenu (ContextMenu.isSupported) | 是 | 是 | 否 | 否 | 否 |

| 类或功能 | 桌面 | extendedDesktop | mobileDevice | tv | extendedTV |
|---|---|---|---|---|---|
| DatagramSocket (DatagramSocket.isSupported) | 是 | 是 | 否 | 否 | 否 |
| DockIcon (NativeApplication.supportsDockIcon) | 检查 | 检查 | 否 | 否 | 否 |
| 拖放 (NativeDragManager.isSupported) | 是 | 是 | 检查 | 否 | 否 |
| EncyptedLocalStore (EncyptedLocalStore.isSupported) | 是 | 是 | 是 | 否 | 否 |
| Flash Access (DRMManager.isSupported) | 是 | 是 | 否 | 是 | 是 |
| GameInput (GameInput.isSupported) | 否 | 否 | 否 | 是 | 是 |
| Geolocation (Geolocation.isSupported) | 否 | 否 | 检查 | 否 | 否 |
| HTMLLoader (HTMLLoader.isSupported) | 是 | 是 | 否 | 否 | 否 |
| IME (IME.isSupported) | 是 | 是 | 检查 | 否 | 否 |
| LocalConnection (LocalConnection.isSupported) | 是 | 是 | 否 | 是 | 是 |
| Microphone (Microphone.isSupported) | 是 | 是 | 检查 | 否 | 否 |
| 多频道音频 (Capabilities.hasMultiChannelAudio()) | 否 | 否 | 否 | 检查 | 检查 |
| 本机扩展 | 否 | 是 | 是 | 否 | 是 |
| NativeMenu (NativeMenu.isSupported) | 是 | 是 | 否 | 否 | 否 |
| NativeProcess (NativeProcess.isSupported) | 否 | 是 | 否 | 否 | 否 |
| NativeWindow (NativeWindow.isSupported) | 是 | 是 | 否 | 否 | 否 |
| NetworkInfo (NetworkInfo.isSupported) | 是 | 是 | 检查 | 是 | 是 |
| 使用默认应用程序打开文件 | 有限制 | 是 | 否 | 否 | 否 |
| PrintJob (PrintJob.isSupported) | 是 | 是 | 否 | 否 | 否 |
| SecureSocket (SecureSocket.isSupported) | 是 | 是 | 否 | 检查 | 检查 |
| ServerSocket (ServerSocket.isSupported) | 是 | 是 | 否 | 否 | 否 |
| Shader | 是 | 是 | 有限制 | 否 | 否 |
| Stage3D (Stage.stage3Ds.length) | 是 | 是 | 否 | 否 | 否 |
| 舞台方向 (Stage.supportsOrientationChange) | 否 | 否 | 是 | 否 | 否 |
| StageVideo | 否 | 否 | 检查 | 是 | 是 |
| StageWebView (StageWebView.isSupported) | 是 | 是 | 是 | 否 | 否 |
| 登录时启动应用程序 (NativeApplication.supportsStartAtLogin) | 是 | 是 | 否 | 否 | 否 |

| 类或功能 | 桌面 | extendedDesktop | mobileDevice | tv | extendedTV |
|---|---|---|---|---|---|
| StorageVolumeInfo (StorageVolumeInfo.isSupported) | 是 | 是 | 否 | 检查 | 检查 |
| 系统空闲模式 | 否 | 否 | 是 | 否 | 否 |
| SystemTrayIcon (NativeApplication.supportsSystemTrayIcon) | 检查 | 检查 | 否 | 否 | 否 |
| Text Layout Framework 输入 | 是 | 是 | 否 | 否 | 否 |
| Updater (Updater.isSupported) | 是 | 否 | 否 | 否 | 否 |
| XMLSignatureValidator (XMLSignatureValidator.isSupported) | 是 | 是 | 否 | 是 | 是 |

表中各项具有以下含义：

- 检查 — 在该配置文件中，某些设备支持该功能，但并非所有设备均支持该功能。在使用该功能之前，应在运行时检查是否支持该功能。
- 有限制 — 支持该功能，但具有一些重要限制。有关更多信息，请参阅相关文档。
- 否 — 该配置文件不支持该功能。
- 是 — 该配置文件支持该功能。请注意，个别计算设备可能缺少某项功能所需的硬件。例如，并非所有手机均带有摄像头。

## 在使用 ADL 进行调试时指定配置文件

**Adobe AIR 2** 和更高版本

ADL 将检查是否在应用程序描述符文件的 supportedProfiles 元素中指定了支持的配置文件。如果已指定，则默认情况下，ADL 在调试时将使用列出的第一个支持的配置文件作为配置文件。

可使用 -profile 命令行参数为 ADL 调试会话指定一个配置文件。（请参阅 第 121 页的 “AIR Debug Launcher (ADL)”。）无论是否在应用程序描述符文件的 supportedProfiles 元素中指定了配置文件，您都可以使用此参数。然而，如果您确实指定了一个 supportedProfiles 元素，则它必须包含您在命令行中指定的配置文件。否则， ADL 将生成错误。

# 第 16 章 : AIR.SWF 浏览器内 API

## 自定义无缝安装 badge.swf

除了使用 SDK 随附的 badge.swf 文件，您还可以创建自己的 SWF 文件以供在浏览器页面中使用。自定义的 SWF 文件可以通过以下方式与运行时进行交互：

- 它可以安装 AIR 应用程序。请参阅第 194 页的 " 从浏览器安装 AIR 应用程序 "。
- 它可以检查是否已安装特定 AIR 应用程序。请参阅第 194 页的 " 从网页检查是否已安装 AIR 应用程序 "。
- 它可以检查是否已安装运行时。请参阅第 193 页的 " 检查是否已安装运行时 "。
- 它可以在用户的系统中启动安装的 AIR 应用程序。请参阅第 195 页的 " 从浏览器启动安装的 AIR 应用程序 "。

所有这些功能都是通过在承载于 adobe.com 上的 SWF 文件 air.swf 中调用 API 提供的。您可以自定义 badge.swf 文件并从自己的 SWF 文件中调用 air.swf API。

另外，在浏览器中运行的 SWF 文件可以通过使用 LocalConnection 类与正在运行的 AIR 应用程序通信。有关详细信息，请参阅与其他 Flash Player 和 AIR 实例通信 （针对 ActionScript 开发人员）或与其他 Flash Player 和 AIR 实例通信 （针对 HTML 开发人员）。

重要说明：本节中所述的功能 （以及 air.swf 文件中的 API）要求最终用户在 Windows 或 Mac OS 的 Web 浏览器中安装 Adobe® Flash® Player 9 更新 3。在 Linux 中，无缝安装功能需要 Flash Player 10（10.0.12.36 版或更高版本）。您可以编写代码来检查已安装的 Flash Player 版本，如果未安装所需的 Flash Player 版本，可以为用户提供替代界面。例如，如果安装的是 Flash Player 的旧版本，您可以提供下载 AIR 文件版本的链接 （而不是使用 badge.swf 文件或 air.swf API 来安装应用程序）。

## 使用 badge.swf 文件安装 AIR 应用程序

AIR SDK 和 Flex SDK 中包含一个 badge.swf 文件，通过此文件，您可轻松使用无缝安装功能。 badge.swf 可以从网页中的链接安装运行时和 AIR 应用程序。为您提供了 badge.swf 文件及其源代码以供您在您的网站上分发。

在网页中嵌入 **badge.swf** 文件

1. 找到下列文件 （这些文件已在 AIR SDK 或 Flex SDK 的 samples/badge 目录中提供），然后将它们添加到您的 Web 服务器。
   - badge.swf
   - default_badge.html
   - AC_RunActiveContent.js
2. 在文本编辑器中打开 default_badge.html 页。
3. 在 default_badge.html 页中的 AC_FL_RunContent() JavaScript 函数中，调整以下参数的 FlashVars 参数定义：

| 参数 | 说明 |
|---|---|
| appname | 应用程序的名称，如果没有安装运行时，则由 SWF 文件显示。 |
| appurl | （必需）。要下载的 AIR 文件的 URL。必须使用绝对 （而非相对） URL。 |
| airversion | （必需）。对于运行时 1.0 版，将此参数设置为 1.0。 |

| 参数 | 说明 |
| --- | --- |
| imageurl | 要在标志中显示的图像（可选）的 URL。 |
| buttoncolor | 下载按钮的颜色（以十六进制值的形式指定，例如 FFCC00）。 |
| messagecolor | 如果没有安装运行时，则为按钮下方显示的文本消息的颜色（以十六进制值的形式指定，如 FFCC00）。 |

4 badge.swf 文件的最小大小为 217 像素宽 x 180 像素高。调整 AC_FL_RunContent() 函数的 width 和 height 参数的值，以满足您的需要。

5 重命名 default_badge.html 文件并调整其代码（或将其包含在另一个 HTML 页中），以满足您的需要。

注：对于加载 badge.swf 文件的 HTML embed 标签，不要设置 wmode 属性；将其保留为默认设置 ("window")。其他 wmode 设置将阻止在某些系统上安装。同样，使用其他 wmode 设置会产生错误："Error #2044: Unhandled ErrorEvent:.text=Error #2074: The stage is too small to fit the download ui."。

您也可以编辑和重新编译 badge.swf 文件。有关详细信息，请参阅第 192 页的“修改 badge.swf 文件”。

## 从网页中的无缝安装链接安装 AIR 应用程序

将无缝安装链接添加到页面中后，用户即可通过在 SWF 文件中单击此链接来安装 AIR 应用程序。

1 在已安装 Flash Player（Windows 和 Mac OS 中为版本 9 更新 3 或更高版本，Linux 中为版本 10）的 Web 浏览器中导航到此 HTML 页。

2 在此网页中，单击 badge.swf 文件中的链接。

- 如果您已安装运行时，请跳至下一步。
- 如果您尚未安装运行时，将显示一个对话框，询问您是否要安装它。安装运行时（请参阅第 2 页的“Adobe AIR 安装”），然后接着执行下一步。

3 在“安装”窗口中，保留默认设置处于选定状态不变，然后单击“继续”。

在 Windows 计算机中， AIR 会自动执行以下操作：

- 将应用程序安装到 c:\Program Files\ 中
- 为此应用程序创建一个桌面快捷方式
- 创建“开始”菜单快捷方式
- 在“添加 / 删除程序”控制面板中添加一个应用程序条目

在 Mac 操作系统中，安装程序会将该应用程序添加到 Applications 目录（例如，添加到 Mac 操作系统中的 /Applications 目录中）。

在 Linux 计算机上， AIR 自动执行以下操作：

- 将应用程序安装到 /opt 中。
- 为此应用程序创建一个桌面快捷方式
- 创建“开始”菜单快捷方式
- 在系统包管理器中加入该应用程序的条目

4 选择所需选项，然后单击“安装”按钮。

5 安装完成后，单击“完成”。

## 修改 badge.swf 文件

Flex SDK 和 AIR SDK 提供了 badge.swf 文件的源文件。这些文件包含在 SDK 的 samples/badge 文件夹中：

| 源文件 | 说明 |
| --- | --- |
| badge.fla | 用于编译 badge.swf 文件的 Flash 源文件。 badge.fla 文件编译成 SWF 9 文件 （可以在 Flash Player 中加载）。 |
| AIRBadge.as | 定义在 basdge.fla 文件中使用的基类的 ActionScript 3.0 类。 |

您可以使用 Flash Professional 重新设计 badge.fla 文件的可视界面。

AIRBadge() 构造函数在 AIRBadge 类中定义，用于加载承载于 http://airdownload.adobe.com/air/browserapi/air.swf 上的 air.swf 文件。 air.swf 文件包含用于使用无缝安装功能的代码。

成功加载 air.swf 文件时，会调用 onInit() 方法 （在 AIRBadge 类中）：

```
private function onInit(e:Event):void {
    _air = e.target.content;
    switch (_air.getStatus()) {
        case "installed" :
            root.statusMessage.text = "";
            break;
        case "available" :
            if (_appName && _appName.length > 0) {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor + "'>In order to run " + _appName +
                        ", this installer will also set up Adobe® AIR®.</font></p>";
            } else {
                root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor + "'>In order to run this application, "
                        + "this installer will also set up Adobe® AIR®.</font></p>";
            }
            break;
        case "unavailable" :
            root.statusMessage.htmlText = "<p align='center'><font color='#"
                        + _messageColor
                        + "'>Adobe® AIR® is not available for your system.</font></p>";
            root.buttonBg_mc.enabled = false;
            break;
    }
}
```

此代码将全局 _air 变量设置为所加载的 air.swf 文件的主类。此类包括以下公共方法， badge.swf 文件通过访问这些方法来调用无缝安装功能：

| 方法 | 说明 |
| --- | --- |
| getStatus() | 确定计算机上是否已安装 （或是否可以安装）运行时。有关详细信息，请参阅第 193 页的 " 检查是否已安装运行时 "。<br>• runtimeVersion - 指示要安装的应用程序所需的运行时版本 （例如 "1.0.M6"）的字符串。 |
| installApplication() | 在用户的计算机上安装指定的应用程序。有关详细信息，请参阅第 194 页的 " 从浏览器安装 AIR 应用程序 "。<br>• url - 定义 URL 的字符串。必须使用绝对 （而非相对） URL 路径。<br>• runtimeVersion — 指示要安装的应用程序所需的运行时版本 （例如 "2.5"）的字符串。<br>• arguments - 要传递给此应用程序的参数 （如果此应用程序在安装后启动）。如果在应用程序描述符文件中将 allowBrowserInvocation 元素设置为 true，则应用程序会在安装后启动。（有关应用程序描述符文件的详细信息，请参阅第 154 页的 "AIR 应用程序描述符文件 "。）如果因从浏览器进行无缝安装而导致应用程序启动 （用户选择在安装后启动），则仅当传递参数时，应用程序的 NativeApplication 对象才调度 BrowserInvokeEvent 对象。请考虑您传递给应用程序的数据存在的安全隐患。有关详细信息，请参阅第 195 页的 " 从浏览器启动安装的 AIR 应用程序 "。 |

url 和 runtimeVersion 的设置通过容器 HTML 页中的 FlashVars 设置传入 SWF 文件。

如果应用程序在安装后自动启动，您可以使用 LocalConnection 通信让已安装的应用程序在调用时与 badge.swf 文件联系。有关详细信息，请参阅与其他 Flash Player 和 AIR 实例通信（针对 ActionScript 开发人员）或与其他 Flash Player 和 AIR 实例通信（针对 HTML 开发人员）。

您也可以调用 air.swf 文件的 getApplicationVersion() 方法来检查是否已安装应用程序。可以在开始应用程序安装过程之前调用此方法，亦可在安装开始之后进行调用。有关详细信息，请参阅第 194 页的“从网页检查是否已安装 AIR 应用程序”。

# 加载 air.swf 文件

您可以创建自己的 SWF 文件，使之使用 air.swf 文件中的 API 从浏览器中的网页与运行时和 AIR 应用程序交互。 air.swf 文件承载于 http://airdownload.adobe.com/air/browserapi/air.swf。若要从 SWF 文件中引用 air.swf API，请将 air.swf 文件加载到 SWF 文件所在的应用程序域中。下面的代码显示了将 air.swf 文件加载到执行加载的 SWF 文件所在的应用程序域中的示例：

```
 var airSWF:Object; // This is the reference to the main class of air.swf
var airSWFLoader:Loader = new Loader(); // Used to load the SWF
var loaderContext:LoaderContext = new LoaderContext();
                                 // Used to set the application domain

loaderContext.applicationDomain = ApplicationDomain.currentDomain;

airSWFLoader.contentLoaderInfo.addEventListener(Event.INIT, onInit);
airSWFLoader.load(new URLRequest("http://airdownload.adobe.com/air/browserapi/air.swf"),
                    loaderContext);

function onInit(e:Event):void
{
    airSWF = e.target.content;
}
```

一旦加载了 air.swf 文件 （Loader 对象的 contentLoaderInfo 对象调度 init 事件时），您就可以调用任何 air.swf API，如下面部分所述。

注：与 AIR SDK 和 Flex SDK 一起提供的 badge.swf 文件将自动加载 air.swf 文件。请参阅第 190 页的“使用 badge.swf 文件安装 AIR 应用程序”。本节中的说明适用于创建您自己的加载 air.swf 文件的 SWF 文件。

# 检查是否已安装运行时

SWF 文件可以通过在从 http://airdownload.adobe.com/air/browserapi/air.swf 加载的 air.swf 文件中调用 getStatus() 方法，检查是否已安装运行时。有关详细信息，请参阅第 193 页的“加载 air.swf 文件”。

加载 air.swf 文件后， SWF 文件便可以调用 air.swf 文件的 getStatus() 方法，如下所示：

```
 var status:String = airSWF.getStatus();
```

getStatus() 方法会根据计算机上运行时的状态，返回下列字符串值之一：

| 字符串值 | 说明 |
| --- | --- |
| "available" | 运行时可以安装在此计算机上，但当前未安装。 |
| "unavailable" | 运行时无法安装在此计算机上。 |
| "installed" | 运行时已安装在此计算机上。 |

如果浏览器中未安装所需的 Flash Player 版本（Windows 和 Mac OS 中为版本 9 更新 3 或更高版本，Linux 中为版本 10），则 getStatus() 方法会引发错误。

# 从网页检查是否已安装 AIR 应用程序

SWF 文件可以通过在从 http://airdownload.adobe.com/air/browserapi/air.swf 加载的 air.swf 文件中调用 getApplicationVersion() 方法，检查是否已安装 AIR 应用程序 （具有匹配的应用程序 ID 和发行商 ID）。有关详细信息，请参阅第 193 页的 “ 加载 air.swf 文件 ”。

加载 air.swf 文件后， SWF 文件便可以调用 air.swf 文件的 getApplicationVersion() 方法，如下所示：

```
 var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";
airSWF.getApplicationVersion(appID, pubID, versionDetectCallback);

function versionDetectCallback(version:String):void
{
    if (version == null)
    {
        trace("Not installed.");
        // Take appropriate actions. For instance, present the user with
        // an option to install the application.
    }
    else
    {
        trace("Version", version, "installed.");
        // Take appropriate actions. For instance, enable the
        // user interface to launch the application.
    }
}
```

getApplicationVersion() 方法具有如下参数：

| 参数 | 说明 |
|---|---|
| appID | 此应用程序的应用程序 ID。有关详细信息，请参阅 第 169 页的 “id”。 |
| pubID | 此应用程序的发行商 ID。有关详细信息，请参阅 第 178 页的 “publisherID”。如果所讨论的应用程序没有发行商 ID，请将 pubID 参数设置为空字符串 (“”)。 |
| callback | 用作处理函数的回调函数。getApplicationVersion() 方法异步运行，在检测到已安装的版本 （或没有已安装的版本）时，会调用此回调方法。回调方法定义必须包含一个参数，此参数为一个字符串，设置为已安装的应用程序的版本字符串。如果未安装此应用程序，则会将一个 null 值传递给此函数，如上一代码示例所示。 |

如果浏览器中未安装所需的 Flash Player 版本（Windows 和 Mac OS 中为版本 9 更新 3 或更高版本，Linux 中为版本 10），则 getApplicationVersion() 方法会引发错误。

注：从 AIR 1.5.3 开始，将弃用发行商 ID。不再自动向应用程序分配发行商 ID。为了向后兼容，应用程序可以继续指定发行商 ID。

# 从浏览器安装 AIR 应用程序

SWF 文件可以通过在从 http://airdownload.adobe.com/air/browserapi/air.swf 加载的 air.swf 文件中调用 installApplication() 方法，安装 AIR 应用程序。有关详细信息，请参阅第 193 页的 “ 加载 air.swf 文件 ”。

加载 air.swf 文件后， SWF 文件便可以调用 air.swf 文件的 installApplication() 方法，如下面的代码中所示：

```
var url:String = "http://www.example.com/myApplication.air";
var runtimeVersion:String = "1.0";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.installApplication(url, runtimeVersion, arguments);
```

installApplication() 方法用于在用户的计算机上安装指定的应用程序。此方法具有以下参数:

| 参数 | 说明 |
|---|---|
| url | 一个字符串，定义要安装的 AIR 文件的 URL。必须使用绝对 （而非相对） URL 路径。 |
| runtimeVersion | 一个字符串，指示要安装的应用程序所需的运行时版本 （例如 “1.0”）。 |
| arguments | 要传递给此应用程序的参数数组 （如果此应用程序在安装后启动）。参数中只能识别字母数字字符。如果需要传递其他值，请考虑使用编码方案。<br><br>如果在应用程序描述符文件中将 allowBrowserInvocation 元素设置为 true，则应用程序会在安装后启动。（有关应用程序描述符文件的详细信息，请参阅第 154 页的 “AIR 应用程序描述符文件 ”。）如果因从浏览器进行无缝安装而导致应用程序启动 （用户选择在安装后启动），则仅当已传递参数时，应用程序的 NativeApplication 对象才调度 BrowserInvokeEvent 对象。有关详细信息，请参阅第 195 页的 “ 从浏览器启动安装的 AIR 应用程序 ”。 |

仅当在用户事件 （例如鼠标单击）的事件处理函数中调用 installApplication() 方法时，此方法才能执行。

如果浏览器中未安装所需的 Flash Player 版本（Windows 和 Mac OS 中为版本 9 更新 3 或更高版本，Linux 中为版本 10），则 installApplication() 方法会引发错误。

在 Mac 操作系统中，若要安装某一应用程序的更新版本，用户必须拥有足够的系统权限才能将新版本安装到应用程序目录中 （如果此应用程序更新运行时，则还须拥有管理权限）。在 Windows 中，用户必须具有管理权限。

您也可以调用 air.swf 文件的 getApplicationVersion() 方法来检查是否已安装应用程序。可以在开始应用程序安装过程之前调用此方法，亦可在安装开始之后进行调用。有关详细信息，请参阅第 194 页的 “ 从网页检查是否已安装 AIR 应用程序 ”。此应用程序运行后便可以通过使用 LocalConnection 类与浏览器中的 SWF 内容通信。有关详细信息，请参阅与其他 Flash Player 和 AIR 实例通信 （针对 ActionScript 开发人员）或与其他 Flash Player 和 AIR 实例通信 （针对 HTML 开发人员）。

# 从浏览器启动安装的 AIR 应用程序

若要使用浏览器调用功能 （使其可以从浏览器启动），目标应用程序的应用程序描述符文件必须包含以下设置:

```
<allowBrowserInvocation>true</allowBrowserInvocation>
```

有关应用程序描述符文件的详细信息，请参阅第 154 页的 “AIR 应用程序描述符文件 ”。

浏览器中的 SWF 文件可以通过在从 http://airdownload.adobe.com/air/browserapi/air.swf 加载的 air.swf 文件中调用 launchApplication() 方法，启动 AIR 应用程序。有关详细信息，请参阅第 193 页的 “ 加载 air.swf 文件 ”。

加载 air.swf 文件后， SWF 文件便可以调用 air.swf 文件的 launchApplication() 方法，如下面的代码所示:

```
var appID:String = "com.example.air.myTestApplication";
var pubID:String = "02D88EEED35F84C264A183921344EEA353A629FD.1";
var arguments:Array = ["launchFromBrowser"]; // Optional
airSWF.launchApplication(appID, pubID, arguments);
```

launchApplication() 方法在 air.swf 文件 （在用户界面 SWF 文件所在的应用程序域中加载）的顶级定义。调用此方法将导致 AIR 启动指定的应用程序 （如果该应用程序已安装，且通过应用程序描述符文件中的 allowBrowserInvocation 设置允许浏览器调用）。此方法具有以下参数:

| 参数 | 说明 |
| --- | --- |
| appID | 要启动的应用程序的应用程序 ID。有关详细信息，请参阅 第 169 页的 “id”。 |
| pubID | 要启动的应用程序的发行商 ID。有关详细信息，请参阅 第 178 页的 “publisherID”。如果所讨论的应用程序没有发行商 ID，请将 pubID 参数设置为空字符串 (“”)。 |
| arguments | 要传递给此应用程序的参数数组。此应用程序的 NativeApplication 对象调度 arguments 属性设置为此数组的 BrowserInvokeEvent 事件。参数中只能识别字母数字字符。如果需要传递其他值，请考虑使用编码方案。 |

仅当在用户事件 （例如鼠标单击）的事件处理程序中调用 launchApplication() 方法时，此方法才执行。

如果浏览器中未安装所需的 Flash Player 版本（Windows 和 Mac OS 中为版本 9 更新 3 或更高版本，Linux 中为版本 10），则 launchApplication() 方法会引发错误。

如果在应用程序描述符文件中将 allowBrowserInvocation 元素设置为 false，则调用 launchApplication() 方法将不起任何作用。

在显示用于启动应用程序的用户界面之前，您可能需要在 air.swf 文件中调用 getApplicationVersion() 方法。有关详细信息，请参阅第 194 页的 “ 从网页检查是否已安装 AIR 应用程序 ”。

当通过浏览器调用功能调用此应用程序时，此应用程序的 NativeApplication 对象将调度 BrowserInvokeEvent 对象。有关详细信息，请参阅从浏览器调用 AIR 应用程序（针对 ActionScript 开发人员）或从浏览器调用 AIR 应用程序（针对 HTML 开发人员）。

如果使用浏览器调用功能，请务必考虑安全性问题。从浏览器调用 AIR 应用程序 （针对 ActionScript 开发人员）和从浏览器调用 AIR 应用程序 （针对 HTML 开发人员）中介绍了这些含义。

此应用程序运行后便可以通过使用 LocalConnection 类与浏览器中的 SWF 内容通信。有关详细信息，请参阅与其他 Flash Player 和 AIR 实例通信（针对 ActionScript 开发人员）或与其他 Flash Player 和 AIR 实例通信（针对 HTML 开发人员）。

注：从 AIR 1.5.3 开始，将弃用发行商 ID。不再自动向应用程序分配发行商 ID。为了向后兼容，应用程序可以继续指定发行商 ID。

# 第 17 章：更新 AIR 应用程序

用户可以通过双击其计算机上的 AIR 文件或从浏览器中（使用无缝安装功能）安装或更新 AIR 应用程序。Adobe® AIR® 安装应用程序将管理此安装，在用户更新现已存在的应用程序时将向其发出警告。

不过，也可以使用 Updater 类让安装的应用程序自行更新到新版本。（安装的应用程序可能检测到有新版本可供下载和安装。）Updater 类包括 update() 方法，此方法让您能够指向用户计算机上的 AIR 文件，并更新为该版本。必须将您的应用程序打包为 AIR 文件，以便使用 Updater 类。打包为本机可执行文件或包的应用程序应使用本机平台所提供的更新设备。

更新 AIR 文件的应用程序 ID 和发布者 ID 必须与要更新的应用程序匹配。发行商 ID 派生自签名证书。必须使用同一证书对更新和要更新的应用程序进行签名。

对于 AIR 1.5.3 或更高版本，应用程序描述符文件包含一个 <publisherID> 元素。如果存在使用 AIR 1.5.2 或更低版本开发的应用程序版本，则必须使用此元素。有关详细信息，请参阅 第 178 页的 “publisherID”。

从 AIR 1.1 及更高版本起，您可以对应用程序进行迁移以使用新的代码签名证书。对应用程序进行迁移以使用新的签名涉及使用新的和原始的证书对更新 AIR 文件进行签名。证书迁移是一个单向过程。迁移后，只有使用新证书 （或同时使用新的和原始证书）进行签名的 AIR 文件才会被识别为对现有安装的更新。

管理应用程序的更新可能非常复杂。 AIR 1.5 包括新的 AdobeAIR 应用程序更新框架。此框架提供的 API 可帮助开发人员在 AIR 应用程序中提供良好的更新功能。

可以使用证书迁移将自签名证书更改为商业代码签名证书，或将一个自签名证书或商业证书更改为另一个自签名证书或商业证书。如果未进行证书迁移，则现有用户必须先删除当前的应用程序版本才能安装新版本。有关详细信息，请参阅第 147 页的 “ 更改证书 ”。

在您的应用程序中包含更新机制是一种好做法。如果创建一个新的应用程序版本，则更新机制可提示用户安装新版本。

AIR 应用程序安装程序会在安装、更新或删除 AIR 应用程序时创建日志文件。可参考这些日志以帮助确定任何安装问题的原因。请参阅安装日志。

注：Adobe AIR 运行时的新版本可能包含 WebKit 的更新版本。 WebKit 的更新版本可能会对已部署的 AIR 应用程序中的 HTML 内容造成意外更改。这些更改可能要求您更新您的应用程序。更新机制可通知用户存在应用程序的新版本。有关详细信息，请参阅关于 HTML 环境 （针对 ActionScript 开发人员）或关于 HTML 环境 （针对 HTML 开发人员）。

## 关于更新应用程序

Updater 类 （在 flash.desktop 包中）包含一个 update() 方法，您可以使用它更新当前运行的具有不同版本的应用程序。例如，如果用户桌面上有某个版本的 AIR 文件 （“Sample_App_v2.air”），则可以使用以下代码更新该应用程序。

ActionScript 示例：

```
var updater:Updater = new Updater();
var airFile:File = File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version:String = "2.01";
updater.update(airFile, version);
```

JavaScript 示例：

```
var updater = new air.Updater();
var airFile = air.File.desktopDirectory.resolvePath("Sample_App_v2.air");
var version = "2.01";
updater.update(airFile, version);
```

在应用程序使用 Updater 类之前，用户或应用程序必须将更新版本的 AIR 文件下载到计算机。有关详细信息，请参阅第 199 页的 “ 将 AIR 文件下载到用户的计算机 ”。

## 调用 Updater.update() 方法的结果

当运行时中的应用程序调用 update() 方法时，运行时将关闭此应用程序，然后尝试从 AIR 文件安装新版本。运行时将检查在 AIR 文件中指定的应用程序 ID 和发布者 ID 是否与调用 update() 方法的应用程序的应用程序 ID 和发布者 ID 相匹配。（有关应用程序 ID 和发布者 ID 的信息，请参阅第 154 页的 “AIR 应用程序描述符文件 ”。）运行时还将检查版本字符串是否与传递给 update() 方法的 version 字符串相匹配。如果安装成功完成，运行时将打开新版本的应用程序。否则 （如果安装无法完成），它将重新打开应用程序的现有 （预安装）版本。

在 Mac OS 中，若要安装某一应用程序的更新版本，用户必须具有足够的系统权限才能将新版本安装到应用程序目录中。在 Windows 和 Linux 中，用户必须具有管理权限。

如果应用程序的更新版本要求运行时的更新版本，则会安装新的运行时版本。若要更新运行时，用户必须具有计算机的管理权限。

在使用 ADL 测试某个应用程序时，调用 update() 方法会导致运行时异常。

## 关于版本字符串

指定为 update() 方法的 version 参数的字符串必须与要安装的 AIR 文件的应用程序描述符文件的 version 或 versionNumber 元素中的字符串相一致。出于安全方面的考虑，需要指定 version 参数。通过要求应用程序验证 AIR 文件中的版本号，该应用程序将不会意外安装旧版本。（应用程序的旧版本可能包含已在当前安装的应用程序中修复的安全漏洞。）应用程序还应检查 AIR 文件中的版本字符串与安装的应用程序中的版本字符串是否相符，以防止降级攻击。

在 AIR 2.5 之前的版本中，版本字符串可以是任何格式。例如，可以是 “2.01” 也可以是 “version 2”。在 AIR 2.5 或更新版本中，版本字符串必须是最多三位数字序列，这三位数字以句号 (.) 分隔。例如， “.0”、 “1.0” 和 “67.89.999” 全部都是有效的版本号。应在更新应用程序之前验证更新版本字符串。

如果 Adobe AIR 应用程序通过 Web 下载 AIR 文件，则运用一种机制使 Web 服务能够通知 AIR 应用程序正在下载的版本是一种很好的做法。然后，应用程序可以将此字符串用作 update() 方法的 version 参数。如果通过其他方式获取 AIR 文件，且 AIR 文件的版本未知，则 AIR 应用程序可以检查 AIR 文件来确定版本信息。（AIR 文件是使用 ZIP 压缩的归档文件，应用程序描述符文件是该归档文件中的第二个记录。）

有关应用程序描述符文件的详细信息，请参阅第 154 页的 “AIR 应用程序描述符文件 ”。

## 对应用程序更新工作流程进行签名

以临时方式发布更新会使管理多个应用程序版本的任务变得复杂，并导致跟踪证书过期日期变得非常困难。证书可能在您发布更新之前就已过期。

Adobe AIR 运行时会将所发布的不带迁移签名的应用程序更新视为新应用程序。用户在安装应用程序更新之前，必须卸载当前的 AIR 应用程序。

为了解决这一问题，请使用最新证书将更新后的每个应用程序上载到单独的部署 URL。并包含一项机制：当您的证书处于 180 天宽限期时，提醒您应用迁移签名。有关更多信息，请参阅第 151 页的 “ 对 AIR 应用程序的更新版本进行签名 ”。

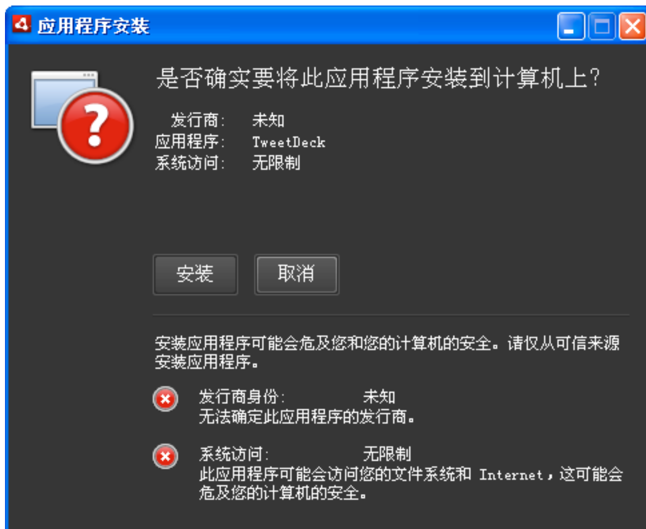
有关如何应用签名的信息，请参阅 第 125 页的 “ADT 命令 ”。

执行以下任务以简化应用迁移签名的过程：

- 将更新后的每个应用程序上载到单独的部署 URL。
- 将升级描述符 XML 文件和更新的最新证书上载到相同的 URL。
- 使用最新证书对更新后的应用程序进行签名。
- 使用用于对位于不同 URL 的上一版本进行签名的证书为更新后的应用程序应用迁移签名。

# 提供自定义应用程序更新用户界面

AIR 包括一个默认的更新界面：




此界面总是在用户首次在计算机上安装某个版本的应用程序时使用。不过，您可以定义自己的界面以供后续实例使用。如果应用程序定义自定义更新界面，请在应用程序描述符文件中为当前安装的应用程序指定一个 customUpdateUI 元素：

```
<customUpdateUI>true</customUpdateUI>
```

当应用程序已经安装，且用户打开应用程序 ID 和发布者 ID 与安装的应用程序相匹配的 AIR 文件时，运行时将打开此应用程序，而不是打开默认的 AIR 应用程序安装程序。有关详细信息，请参阅 第 163 页的 “customUpdateUI”。

应用程序可以决定何时运行 （当 NativeApplication.nativeApplication 对象调度 load 事件时），以及是否更新应用程序 （使用 Updater 类）。如果决定进行更新，则它会向用户提供自己的安装界面 （不同于标准的运行界面）。

# 将 AIR 文件下载到用户的计算机

为了使用 Updater 类，用户或应用程序必须先在本地将 AIR 文件保存到用户的计算机。

注：AIR 1.5 包括一个更新框架，该框架可帮助开发人员在 AIR 应用程序中提供良好的更新功能。使用此框架比直接使用 Updater 类的 update() 方法简单许多。有关详细信息，请参阅第 202 页的 “ 使用更新框架 ”。

以下代码从一个 URL (http://example.com/air/updates/Sample_App_v2.air) 读取 AIR 文件，然后将该 AIR 文件保存到应用程序存储目录中。

ActionScript 示例：

```
var urlString:String = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq:URLRequest = new URLRequest(urlString);
var urlStream:URLStream = new URLStream();
var fileData:ByteArray = new ByteArray();
urlStream.addEventListener(Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event:Event):void {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile():void {
    var file:File = File.applicationStorageDirectory.resolvePath("My App v2.air");
    var fileStream:FileStream = new FileStream();
    fileStream.open(file, FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

JavaScript 示例：

```
var urlString = "http://example.com/air/updates/Sample_App_v2.air";
var urlReq = new air.URLRequest(urlString);
var urlStream = new air.URLStream();
var fileData = new air.ByteArray();
urlStream.addEventListener(air.Event.COMPLETE, loaded);
urlStream.load(urlReq);

function loaded(event) {
    urlStream.readBytes(fileData, 0, urlStream.bytesAvailable);
    writeAirFile();
}

function writeAirFile() {
    var file = air.File.desktopDirectory.resolvePath("My App v2.air");
    var fileStream = new air.FileStream();
    fileStream.open(file, air.FileMode.WRITE);
    fileStream.writeBytes(fileData, 0, fileData.length);
    fileStream.close();
    trace("The AIR file is written.");
}
```

有关详细信息，请参阅：

- 读取和写入文件的工作流程 （针对 ActionScript 开发人员）
- 读取和写入文件的工作流程 （针对 HTML 开发人员）

# 检查应用程序是否为首次运行

更新应用程序后，可能需要向用户提供 " 快速入门 " 或 " 欢迎 " 消息。在启动时，应用程序将检查是否为首次运行，以便可以确定是否显示此类消息。

注：AIR 1.5 包括一个更新框架，该框架可帮助开发人员在 AIR 应用程序中提供良好的更新功能。此框架提供简单的方法，以检查应用程序的版本是否为首次运行。有关详细信息，请参阅第 202 页的 " 使用更新框架 "。

实现此操作的方法之一是在初始化应用程序时，在应用程序存储目录中保存一个文件。应用程序每次启动时，都会检查该文件是否存在。如果此文件不存在，则表示当前用户首次运行该应用程序。如果此文件已存在，则表示该应用程序至少已运行过一次。如果此文件已存在且包含比当前版本号旧的版本号，则可知该用户是首次运行此新版本。

下面的 Flex 示例对该概念进行了演示：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    title="Sample Version Checker Application"
    applicationComplete="system extension()">
    <mx:Script>
        <![CDATA[
            import flash.filesystem.*;
            public var file:File;
            public var currentVersion:String = "1.2";
            public function system extension():void {
                file = File.applicationStorageDirectory;
                file = file.resolvePath("Preferences/version.txt");
                trace(file.nativePath);
                if(file.exists) {
                    checkVersion();
                } else {
                    firstRun();
                }
            }
            private function checkVersion():void {
                var stream:FileStream = new FileStream();
                stream.open(file, FileMode.READ);
                var reversion:String = stream.readUTFBytes(stream.bytesAvailable);
                stream.close();
                if (reversion != currentVersion) {
                    log.text = "You have updated to version " + currentVersion + ".\n";
                } else {
                    saveFile();
                }
                log.text += "Welcome to the application.";
            }
            private function firstRun():void {
                log.text = "Thank you for installing the application. \n"
                    + "This is the first time you have run it.";
                saveFile();
            }
            private function saveFile():void {
                var stream:FileStream = new FileStream();
                stream.open(file, FileMode.WRITE);
                stream.writeUTFBytes(currentVersion);
                stream.close();
            }
        ]]>
    </mx:Script>
    <mx:TextArea ID="log" width="100%" height="100%" />
</mx:WindowedApplication>
```

以下示例说明了 JavaScript 中的概念：

```
<html>
    <head>
        <script src="AIRAliases.js" />
        <script>
            var file;
            var currentVersion = "1.2";
            function system extension() {
                file = air.File.appStorageDirectory.resolvePath("Preferences/version.txt");
                air.trace(file.nativePath);
                if(file.exists) {
                    checkVersion();
                } else {
                    firstRun();
                }
            }
            function checkVersion() {
                var stream = new air.FileStream();
                stream.open(file, air.FileMode.READ);
                var reversion = stream.readUTFBytes(stream.bytesAvailable);
                stream.close();
                if (reversion != currentVersion) {
                    window.document.getElementById("log").innerHTML
                            = "You have updated to version " + currentVersion + ".\n";
                } else {
                    saveFile();
                }
                window.document.getElementById("log").innerHTML
                                 += "Welcome to the application.";
            }
            function firstRun() {
                window.document.getElementById("log").innerHTML
                            = "Thank you for installing the application. \n"
                            + "This is the first time you have run it.";
                saveFile();
            }
            function saveFile() {
                var stream = new air.FileStream();
                stream.open(file, air.FileMode.WRITE);
                stream.writeUTFBytes(currentVersion);
                stream.close();
            }
        </script>
    </head>
    <body onLoad="system extension()">
        <textarea ID="log" rows="100%" cols="100%" />
    </body>
</html>
```

如果应用程序将数据保存到本地 （例如，保存到应用程序存储目录中），则您可能希望在首次运行时检查以前保存的所有数据（来自以前的版本）。

# 使用更新框架

管理应用程序的更新可能非常繁琐。AdobeAIR 应用程序的更新框架 提供了相关 API，开发人员可以利用这些 API 在 AIR 应用程序中提供强大的更新功能。 AIR 更新框架为开发人员执行以下任务：

- 按指定时间间隔定期检查更新，或者在用户发出请求时检查更新
- 从 Web 来源下载 AIR 文件 （更新）

- 在首次运行新安装的版本时向用户发出警告
- 确认用户希望检查更新
- 向用户显示有关新的更新版本的信息
- 向用户显示下载进度和错误信息

AIR 更新框架为您的应用程序提供了一个示例用户界面。该示例用户界面为用户提供了应用程序更新的基本信息和配置选项。您的应用程序也可以定义一个自定义用户界面，与更新框架配合使用。

AIR 更新框架允许您将有关 AIR 应用程序更新版本的信息存储在简单 XML 配置文件中。对于大多数应用程序，设置这些配置文件以包括基本代码可以为最终用户提供良好的更新功能。

即使不使用更新框架， AIR 应用程序仍可以使用 Adobe AIR 包括的 Updater 类升级到新的版本。通过使用 Updater 类，应用程序可以升级到用户计算机上的 AIR 文件中包含的版本。然而，升级管理可不只是进行基于本地存储的 AIR 文件的应用程序更新。

## AIR 更新框架文件

AIR 更新框架包含在 AIR 2 SDK 的 frameworks/libs/air 目录中。它包括下列文件：

- applicationupdater.swc — 定义更新库的基本功能，供在 ActionScript 中使用。此版本不包含任何用户界面。
- applicationupdater.swf — 定义更新库的基本功能，供在 JavaScript 中使用。此版本不包含任何用户界面。
- applicationupdater_ui.swc — 定义 Flex 4 版本的更新库的基本功能，包括应用程序可以用于显示更新选项的用户界面。
- applicationupdater_ui.swf — 定义 JavaScript 版本的更新库的基本功能，包括应用程序可以用于显示更新选项的用户界面。

有关详细信息，请参阅以下这些章节：

- 第 203 页的 “ 设置 Flex 开发环境 ”
- 第 203 页的 “ 在基于 HTML 的 AIR 应用程序中包括框架文件 ”
- 第 204 页的 “ 基本示例：使用 ApplicationUpdaterUI 版本 ”

## 设置 Flex 开发环境

AIR 2 SDK 的 frameworks/libs/air 目录中的 SWC 文件定义可以在 Flex 和 Flash 开发中使用的类。

若要在使用 Flex SDK 进行编译时使用更新框架，请在对 amxmlc 编译器的调用中包括 ApplicationUpdater.swc 或 ApplicationUpdater_UI.swc 文件。在以下示例中，编译器会加载 Flex SDK 目录的 lib 子目录中的 ApplicationUpdater.swc 文件：

```
amxmlc -library-path+=lib/ApplicationUpdater.swc  -- myApp.mxml
```

在以下示例中，编译器会加载 Flex SDK 目录的 lib 子目录中的 ApplicationUpdater_UI.swc 文件：

```
amxmlc -library-path+=lib/ApplicationUpdater_UI.swc  -- myApp.mxml
```

使用 Flash Builder 进行开发时，在 “ 属性 ” 对话框中 “Flex Build 路径设置 ” 的 “ 库路径 ” 选项卡中添加 SWC 文件。

请确保将 SWC 文件复制到将在 amxmlc 编译器 （使用 Flex SDK）或 Flash Builder 中引用的目录。

## 在基于 HTML 的 AIR 应用程序中包括框架文件

更新框架的 frameworks/html 目录包括以下这些文件：

- applicationupdater.swf — 定义更新库的基本功能，不包括任何用户界面
- applicationupdater_ui.swf — 定义更新库的基本功能，包括应用程序可以用于显示更新选项的用户界面

AIR 应用程序中的 JavaScript 代码可以使用在 SWF 文件中定义的类。

若要使用更新框架，请在应用程序目录 （或子目录）中加入 applicationupdater.swf 或 applicationupdater_ui.swf 文件。然后，在要使用该框架的 HTML 文件中 （在 JavaScript 代码中）添加加载该文件的 script 标记：

```
<script src="applicationUpdater.swf" type="application/x-shockwave-flash"/>
```

或者，使用此 script 标记加载 applicationupdater_ui.swf 文件：

```
<script src="applicationupdater_ui.swf" type="application/x-shockwave-flash"/>
```

这两个文件中定义的 API 将在本文档的其余部分进行描述。

## 基本示例：使用 ApplicationUpdaterUI 版本

更新框架的 ApplicationUpdaterUI 版本提供了可以在应用程序中轻松使用的基本界面。下面是一个基本示例。

首先，创建调用更新框架的 AIR 应用程序：

1 如果您的应用程序是基于 HTML 的 AIR 应用程序，则会加载 applicationupdaterui.swf 文件：

```
<script src="ApplicationUpdater_UI.swf" type="application/x-shockwave-flash"/>
```

2 在 AIR 应用程序程序逻辑中，实例化 ApplicationUpdaterUI 对象。

在 ActionScript 中，使用以下代码：

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

在 JavaScript 中，使用以下代码：

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

您可能希望在加载应用程序后执行的初始化函数中添加此代码。

3 创建名为 updateConfig.xml 的文本文件并在其中添加以下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
    <url>http://example.com/updates/update.xml</url>
    <delay>1</delay>
</configuration>
```

编辑 updateConfig.xml 文件的 URL 元素以与 Web 服务器上的更新描述符文件的最终位置相一致 （请参阅下一过程）。

delay 是应用程序在检查更新之间等待的天数。

4 将 updateConfig.xml 文件添加到 AIR 应用程序的项目目录中。

5 使 updater 对象引用 updateConfig.xml 文件，并调用该对象的 initialize() 方法。

在 ActionScript 中，使用以下代码：

```
appUpdater.configurationFile = new File("app:/updateConfig.xml");
appUpdater.initialize();
```

在 JavaScript 中，使用以下代码：

```
appUpdater.configurationFile = new air.File("app:/updateConfig.xml");
appUpdater.initialize();
```

6 创建另一版本的 AIR 应用程序，该版本与第一个应用程序的版本不同。（该版本将在应用程序描述符文件的 version 元素中指定。）

随后，将更新版本的 AIR 应用程序添加到 Web 服务器：

1 将更新版本的 AIR 文件置于 Web 服务器上。

2 创建名为 updateDescriptor.2.5.xml 的文本文件并在其中添加以下内容：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
       <versionNumber>1.1</versionNumber>
       <url>http://example.com/updates/sample_1.1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

编辑 updateDescriptor.xml 文件中的 versionNumber、 URL 和 description 以便与更新 AIR 文件相匹配。使用 AIR 2.5 SDK （和更新版本）提供的更新框架的应用程序将使用此更新描述符格式。

3 创建名为 updateDescriptor.1.0.xml 的文本文件并在其中添加以下内容：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1</version>
       <url>http://example.com/updates/sample_1.1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

编辑 updateDescriptor.xml 文件中的 version、URL 和 description 以便与更新 AIR 文件相匹配。使用 AIR 2 SDK（和先前版本）提供的更新框架的应用程序将使用此更新描述符格式。

注：仅当您支持在 AIR 2.5 之前创建的应用程序更新时，才需要创建上面第二个更新描述符文件。

4 将 updateDescriptor.2.5.xml 和 updateDescriptor.1.0.xml 文件添加到包含更新 AIR 文件的同一个 Web 服务器目录。

这是一个基本示例，但它提供的更新功能可以满足许多应用程序的需要。本文档的其余部分介绍了如何使用更新框架以最好地满足您的需要。

有关使用更新框架的其他示例，请参阅 Adobe AIR 开发人员中心中的以下示例应用程序：

- 在基于 Flex 的应用程序中更新框架 (http://www.adobe.com/go/learn_air_qs_update_framework_flex_cn)
- 在基于 Flash 的应用程序中更新框架 (http://www.adobe.com/go/learn_air_qs_update_framework_flash_cn)
- 在基于 HTML 的应用程序中更新框架 (http://www.adobe.com/go/learn_air_qs_update_framework_html_cn)

## 更新到 AIR 2.5

因为用于指定应用程序版本号的规则在 AIR 2.5 中已发生更改，AIR 2 更新框架无法分析 AIR 2.5 应用程序描述符中的版本信息。这种不兼容性意味着在更新应用程序以使用 AIR 2.5 SDK 之前，必须更新应用程序以使用新的更新框架。因此，将应用程序从 AIR 2.5 之前的任何版本更新到 AIR 2.5 或更新版本需要两次更新。第一次更新必须使用 AIR 2 命名空间并加入 AIR 2.5 更新框架库（您仍可以使用 AIR 2.5 SDK 创建应用程序包）。第二次更新可以使用 AIR 2.5 命名空间以及加入应用程序的新功能。

您也可以在进行中间更新时，除了使用 AIR Updater 类直接更新到 AIR 2.5 应用程序之外，使其不执行任何其他操作。

下面的示例演示如何将应用程序从版本 1.0 更新到 2.0。版本 1.0 使用旧的 2.0 命名空间。版本 2.0 使用 2.5 命名空间，并具有使用 AIR 2.5 API 实现的新功能。

1 基于应用程序版本 1.0 创建应用程序的中间版本 1.0.1。

  a 在创建应用程序时使用 AIR 2.5 Application Updater 框架。

    注：对于基于 Flash 技术的 AIR 应用程序，使用 applicationupdater.swc 或 applicationupdater_ui.swc ；对于基于 HTML 的 AIR 应用程序，使用 applicationupdater.swf 或 applicationupdater_ui.swf。

  b 使用旧的命名空间和版本为版本 1.0.1 创建更新描述符文件，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
    <update xmlns="http://ns.adobe.com/air/framework/update/description/2.0">
        <version>1.0.1</version>
        <url>http://example.com/updates/sample_1.0.1.air</url>
        <description>This is the intermediate version.</description>
    </update>
```

**2** 使用 AIR 2.5 API 和 2.5 命名空间创建应用程序版本 2.0。

**3** 创建一个更新描述符文件，将应用程序从 1.0.1 版本更新到 2.0 版本。

```
<?xml version="1.0" encoding="utf-8"?>
    <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
        <version>2.0</version>
        <url>http://example.com/updates/sample_2.0.air</url>
        <description>This is the intermediate version.</description>
    </update>
```

## 定义更新描述符文件并将 AIR 文件添加到 Web 服务器

在使用 AIR 更新框架时，您可以在存储于 Web 服务器的更新描述符文件中定义关于可用更新的基本信息。更新描述符文件是简单的 XML 文件。包括在应用程序中的更新框架可对此文件进行检查，以查看是否已上载新的版本。

AIR 2.5 中的更新描述符文件格式已更改。新的格式使用不同的命名空间。原来的命名空间是 "http://ns.adobe.com/air/framework/update/description/1.0"。 AIR 2.5 命名空间是 "http://ns.adobe.com/air/framework/update/description/2.5"。

在 AIR 2.5 之前创建的 AIR 应用程序只能读取 1.0 版更新描述符。使用 AIR 2.5 或更新版本中包括的 updater 框架创建的 AIR 应用程序只能读取 2.5 版更新描述符。由于此版本不兼容性，您通常需要创建两个更新描述符文件。应用程序的 AIR 2.5 版本中的更新逻辑必须下载使用新格式的更新描述符。 AIR 应用程序的先前版本必须继续使用原来的格式。每次发行更新时，必须同时修改这两个文件 （直到不再支持在 AIR 2.5 之前创建的版本为止）。

更新描述符文件包含以下数据：

- versionNumber — AIR 应用程序的新版本。在用于更新 AIR 2.5 应用程序的更新描述符中使用 versionNumber 元素。该值必须与新 AIR 应用程序描述符文件的 versionNumber 元素中使用的字符串相同。如果更新描述符文件中的版本号与更新 AIR 文件的版本号不一致，则更新框架将引发异常。
- version — AIR 应用程序的新版本。使用用于更新在 AIR 2.5 之前创建的应用程序的更新描述符中的 version 元素。该值必须与新 AIR 应用程序描述符文件的 version 元素中使用的字符串相同。如果更新描述符文件中的版本与更新 AIR 文件的版本不一致，则更新框架将引发异常。
- versionLabel — 要向用户显示的人工可读版本字符串。 versionLabel 是可选的，但是只能在 2.5 版更新描述符文件中指定。如果在应用程序描述符中使用 versionLabel 并将其设为相同的值，则使用该字符串。
- url — 更新 AIR 文件的位置。此文件包含更新版本的 AIR 应用程序。
- description — 有关新版本的详细信息。在更新过程中可以向用户显示此信息。

version 和 url 元素是必需的，而 description 元素为可选元素。

下面是一个 2.5 示例版更新描述符文件：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
       <versionNumber>1.1.1</versionNumber>
       <url>http://example.com/updates/sample_1.1.1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

并且，下面是一个示例 1.0 版更新描述符文件：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1.1</version>
       <url>http://example.com/updates/sample_1.1.1.air</url>
       <description>This is the latest version of the Sample application.</description>
    </update>
```

如果您希望使用多种语言定义 description 标记，请使用定义 lang 属性的多个 text 元素：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/2.5">
       <versionNumber>1.1.1</versionNumber>
       <url>http://example.com/updates/sample_1.1.1.air</url>
       <description>
           <text xml:lang="en">English description</text>
           <text xml:lang="fr">French description</text>
           <text xml:lang="ro">Romanian description</text>
       </description>
    </update>
```

将更新描述符文件随更新 AIR 文件一起放置在 Web 服务器上。

更新描述符随附的模板目录包括示例更新描述符文件。这些文件包括单语言和多语言版本。

## 实例化 updater 对象

在代码中加载 AIR 更新框架（请参阅第 203 页的 " 设置 Flex 开发环境 " 和第 203 页的 " 在基于 HTML 的 AIR 应用程序中包括框架文件 "）后，您需要实例化 updater 对象，如以下示例所示。

ActionScript 示例：

```
var appUpdater:ApplicationUpdater = new ApplicationUpdater();
```

JavaScript 示例：

```
var appUpdater = new runtime.air.update.ApplicationUpdater();
```

以上代码使用了 ApplicationUpdater 类（没有提供任何用户界面）。如果要使用提供用户界面的 ApplicationUpdaterUI 类，请使用以下语句。

ActionScript 示例：

```
var appUpdater:ApplicationUpdaterUI = new ApplicationUpdaterUI();
```

JavaScript 示例：

```
var appUpdater = new runtime.air.update.ApplicationUpdaterUI();
```

本文档中的其余代码示例假定您已实例化名为 appUpdater 的 updater 对象。

## 配置更新设置

ApplicationUpdater 和 ApplicationUpdaterUI 可以通过应用程序随附的配置文件，或者通过应用程序中的 ActionScript 或 JavaScript 进行配置。

### 在 XML 配置文件中定义更新设置

更新配置文件为 XML 文件。它可能包含以下元素：

- updateURL — 字符串。表示更新描述符在远程服务器上的位置。允许使用任何有效的 URLRequest 位置。您必须定义 updateURL 属性，可以通过配置文件也可以通过脚本进行定义（请参阅第 206 页的 " 定义更新描述符文件并将 AIR 文件添

加到 Web 服务器 ”）。只有在首先定义此属性后才能使用 updater （在调用 updater 对象的 initialize() 方法之前，在第 210 页的 “ 初始化更新框架 ” 中进行了说明）。

- delay — 一个数字。表示以天为单位表示用于检查更新的时间间隔 （允许使用 0.25 等数值）。值 0 （这是默认值）指定该 updater 不执行自动定期检查。

ApplicationUpdaterUI 的配置文件除了包含 updateURL 和 delay 元素之外，还可包含以下元素：

- defaultUI：dialog 元素的列表。每个 dialog 元素都具有与用户界面中的对话框对应的 name 属性。每个 dialog 元素都具有定义该对话框是否可见的 visible 属性。默认值为 True。 name 属性的可能值如下所示：
  - "checkForUpdate" — 与 “ 检查更新 ”、 “ 没有更新 ” 和 “ 更新错误 ” 对话框相对应
  - "downloadUpdate" — 与 “ 下载更新 ” 对话框相对应
  - "downloadProgress" — 与 “ 下载进度 ” 和 “ 下载错误 ” 对话框相对应
  - "installUpdate" — 与 “ 安装更新 ” 对话框相对应
  - "fileUpdate" — 与 “ 文件更新 ”、 “ 文件没有更新 ” 和 “ 文件错误 ” 对话框相对应
- "unexpectedError" — 与 “ 意外错误 ” 对话框相对应

  在设置为 false 时，对应的对话框不作为更新过程的一部分进行显示。

下面是 ApplicationUpdater 框架的配置文件的一个示例：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
      <url>http://example.com/updates/update.xml</url>
      <delay>1</delay>
</configuration>
```

下面是 ApplicationUpdaterUI 框架的配置文件的一个示例，其中包括 defaultUI 元素的定义：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://ns.adobe.com/air/framework/update/configuration/1.0">
      <url>http://example.com/updates/update.xml</url>
      <delay>1</delay>
      <defaultUI>
         <dialog name="checkForUpdate" visible="false" />
         <dialog name="downloadUpdate" visible="false" />
         <dialog name="downloadProgress" visible="false" />
      </defaultUI>
</configuration>
```

将 configurationFile 属性指定为该文件的位置：

ActionScript 示例：

```
appUpdater.configurationFile = new File("app:/cfg/updateConfig.xml");
```

JavaScript 示例：

```
appUpdater.configurationFile = new air.File("app:/cfg/updateConfig.xml");
```

更新框架的模板目录包括一个示例配置文件 config-template.xml。

## 定义更新设置 ActionScript 或 JavaScript 代码

这些配置参数还可以使用应用程序中的代码进行设置，如下所示：

```
appUpdater.updateURL = " http://example.com/updates/update.xml";
appUpdater.delay = 1;
```

updater 对象的属性为 updateURL 和 delay。这些属性所定义的配置与配置文件中的 updateURL 和 delay 元素相同：更新描述符文件的 URL 以及检查更新的时间间隔。如果您在代码中指定配置文件和 设置，则使用代码设置的任意属性优先于配置文件中相应的设置。

只有在定义 updateURL 属性之后，可以通过配置文件也可以通过脚本进行定义 （请参阅第 206 页的 " 定义更新描述符文件并将 AIR 文件添加到 Web 服务器 "）才能使用 updater（在调用 updater 对象的 initialize() 方法之前，在第 210 页的 " 初始化更新框架 " 中进行了说明）。

ApplicationUpdaterUI 框架定义 updater 对象的以下这些附加属性：

- isCheckForUpdateVisible — 与 " 检查更新 "、 " 没有更新 " 和 " 更新错误 " 对话框相对应
- isDownloadUpdateVisible — 与 " 下载更新 " 对话框相对应
- isDownloadProgressVisible — 与 " 下载进度 " 和 " 下载错误 " 对话框相对应
- isInstallUpdateVisible — 与 " 安装更新 " 对话框相对应
- isFileUpdateVisible — 与 " 文件更新 "、 " 文件没有更新 " 和 " 文件错误 " 对话框相对应
- isUnexpectedErrorVisible — 与 " 意外错误 " 对话框相对应

每个属性都对应于 ApplicationUpdaterUI 用户界面中的一个或多个对话框。每个属性均为布尔值，默认值为 True。在设置为 False 时，对应的对话框不作为更新过程的一部分进行显示。

这些对话框属性优先于更新配置文件中的设置。

## 更新过程

AIR 更新框架可按以下步骤完成更新过程：

1 updater 初始化检查更新检查是否已在定义的延迟间隔内执行 （请参阅第 207 页的 " 配置更新设置 "）。如果更新检查已达到定义的时间，则更新过程继续进行。

2 updater 下载并解释更新描述符文件。

3 updater 下载更新 AIR 文件。

4 updater 安装应用程序的更新版本。

updater 对象在完成每个步骤时对事件进行调度。在 ApplicationUpdater 版本中，可以取消指示过程中某个步骤成功完成的事件。如果取消其中一个事件，则过程中的下一步也会被取消。在 ApplicationUpdaterUI 版本中，updater 提供了允许用户取消或继续进行过程中每个步骤的对话框。

如果取消该事件，则可以调用 updater 对象的方法继续进行此过程。

由于 updater 的 ApplicationUpdater 版本是通过更新过程执行的，因此它在 currentState 属性中记录了其当前的状态。此属性设置为具有以下可能值的字符串：

- "UNINITIALIZED" — updater 尚未初始化。
- "INITIALIZING" — updater 正在初始化。
- "READY" — updater 已初始化。
- "BEFORE_CHECKING" — 尚未检查 updater 中是否有更新描述符文件。
- "CHECKING" — updater 正在检查是否有更新描述符文件。
- "AVAILABLE" — updater 描述符文件可用。
- "DOWNLOADING" — updater 正在下载 AIR 文件。
- "DOWNLOADED" — updater 已下载 AIR 文件。
- "INSTALLING" — updater 正在安装 AIR 文件。

- "PENDING_INSTALLING" — updater 已初始化，但存在未处理的更新。

updater 对象的某些方法仅在 updater 处于特定的状态下才执行。

## 初始化更新框架

在设置配置属性后 （请参阅第 204 页的 " 基本示例：使用 ApplicationUpdaterUI 版本 "），调用 initialize() 方法对更新进行初始化：

```
appUpdater.initialize();
```

此方法执行以下操作：

- 它将初始化更新框架，以静默方式同步安装所有未处理的更新。在应用程序启动过程中需要调用此方法，因为调用此方法时可以重新启动应用程序。
- 它检查是否有被推迟的更新，如果有将安装该更新。
- 如果在更新过程中出现错误，则该方法会从应用程序存储区域中清除更新文件和版本信息。
- 如果延迟已到期，则该方法会启动更新过程。否则，将重新启动计时器。

调用此方法可能会导致 updater 对象调度以下事件：

- UpdateEvent.INITIALIZED — 初始化完成时调度此事件。
- ErrorEvent.ERROR — 初始化过程中出现错误时调度此事件。

在调度 UpdateEvent.INITIALIZED 事件时，更新过程已完成。

在调用 initialize() 方法时，updater 会根据计时器延迟设置启动更新过程并完成所有步骤。但还可以通过调用 updater 对象的 checkNow() 方法随时启动更新过程：

```
appUpdater.checkNow();
```

如果更新过程已在运行，则此方法不执行任何操作。否则，将启动更新过程。

由于调用 checkNow() 方法，会引发 updater 对象调度以下事件：

- UpdateEvent.CHECK_FOR_UPDATE 事件仅在尝试下载更新描述符文件之前调度。

如果取消 checkForUpdate 事件，则可以调用 updater 对象的 checkForUpdate() 方法。（请参阅下一节。）如果不取消该事件，则更新过程会继续检查更新描述符文件。

## 管理 ApplicationUpdaterUI 版本中的更新过程

在 ApplicationUpdaterUI 版本中，用户可以通过用户界面的对话框中的 " 取消 " 按钮来取消过程。此外，可以通过调用 ApplicationUpdaterUI 对象的 cancelUpdate() 方法，以编程方式取消更新过程。

可以设置 ApplicationUpdaterUI 对象的属性或定义更新配置文件中的元素来指定 updater 显示哪些对话框确认。有关详细信息，请参阅第 207 页的 " 配置更新设置 "。

## 管理 ApplicationUpdater 版本中的更新过程

可以调用 ApplicationUpdater 对象调度的事件对象的 preventDefault() 方法来取消更新过程的步骤 （请参阅第 209 页的 " 更新过程 "）。取消默认行为让您的应用程序有机会向用户显示消息，询问其是否要继续。

以下部分将介绍在取消过程的某个步骤后，如何继续进行更新过程。

### 下载并解释更新描述符文件

在更新过程开始之前 ApplicationUpdater 对象调度 checkForUpdate 事件，紧接着 updater 就会尝试下载更新描述符文件。如果取消 checkForUpdate 事件的默认行为，则 updater 将不下载更新描述符文件。可以调用 checkForUpdate() 方法恢复更新过程：

```
appUpdater.checkForUpdate();
```

调用 checkForUpdate() 方法会导致 updater 异步下载和解释更新描述符文件。作为调用 checkForUpdate() 方法的结果，updater 对象可以调度以下事件：

- StatusUpdateEvent.UPDATE_STATUS — updater 已成功下载并解释更新描述符文件时调度此事件。此事件具有以下属性：
  - available — 一个布尔值。如果存在不同于当前应用程序的可用版本，则设置为 true；否则 （版本相同），设置为 false。
  - version — 一个字符串。更新文件的应用程序描述符文件的版本
  - details — 一个数组。如果没有本地化版本的描述，则此数组会返回一个空字符串 ("") 作为第一个元素，而将描述作为第二个元素。

    如果存在多个版本的描述 （位于更新描述符文件中），则该数组包含多个子数组。每个数组包含两个元素：第一个元素为语言代码 （例如 "en"），第二个元素为该语言的相应描述 （一个字符串）。请参阅第 206 页的 " 定义更新描述符文件并将 AIR 文件添加到 Web 服务器 "。
- StatusUpdateErrorEvent.UPDATE_ERROR — 存在错误，而且 updater 无法下载或解释更新描述符文件时调度此事件。

### 下载更新 AIR 文件

updater 成功下载并解释更新描述符文件之后， ApplicationUpdater 对象将调度 updateStatus 事件。默认行为是开始下载更新文件 （如果可用）。如果取消默认行为，则可以调用 downloadUpdate() 方法恢复更新过程：

```
appUpdater.downloadUpdate();
```

调用此方法会导致 updater 异步下载 AIR 文件的更新版本。

downloadUpdate() 方法可以调度以下事件：

- UpdateEvent.DOWNLOAD_START — 建立到服务器的连接时调度此事件。在使用 ApplicationUpdaterUI 库时，此事件将显示带有进度栏的对话框以对下载进度进行跟踪。
- ProgressEvent.PROGRESS — 根据文件下载进度定期调度此事件。
- DownloadErrorEvent.DOWNLOAD_ERROR — 如果在连接或下载更新文件时出现错误，则调度此事件。 HTTP 状态无效时也会调度此事件 （例如 "404 - File not found" （404 - 找不到文件））。此事件具有 errorID 属性，该属性为定义其他错误信息的整数。此外，还具有另一个属性 subErrorID，该属性可能包含更多错误信息。
- UpdateEvent.DOWNLOAD_COMPLETE — updater 已成功下载并解释更新描述符文件时调度此事件。如果不取消此事件，则 ApplicationUpdater 版本会继续安装更新版本。在 ApplicationUpdaterUI 版本中，向用户显示为其提供选项的对话框以继续操作。

### 更新应用程序

更新文件下载完毕后， ApplicationUpdater 对象将调度 downloadComplete 事件。如果取消默认行为，则可以调用 installUpdate() 方法恢复更新过程：

```
appUpdater.installUpdate(file);
```

调用此方法会导致 updater 安装 AIR 文件的更新版本。此方法包括一个参数 file，该参数是引用要用作更新的 AIR 文件的 File 对象。

作为调用 installUpdate() 方法的结果， ApplicationUpdater 对象可以调度 beforeInstall 事件：

- UpdateEvent.BEFORE_INSTALL — 仅在安装更新前调度此事件。有时，阻止目前更新的安装非常有用，这样用户可以完成当前的工作，然后再继续进行更新。调用 Event 对象的 preventDefault() 方法可推迟安装直至下次重新启动，且可以不启动任何其他更新过程。（这些更新包括通过调用 checkNow() 方法或由于定期检查而产生的更新。）

## 从任意 AIR 文件进行安装

可以调用 installFromAIRFile() 方法，安装要从用户计算机上的 AIR 文件进行安装的更新版本：

```
appUpdater.installFromAIRFile();
```

此方法会导致 updater 从 AIR 文件安装应用程序的更新版本。

installFromAIRFile() 方法可以调度以下事件：

- StatusFileUpdateEvent.FILE_UPDATE_STATUS — 在 ApplicationUpdater 成功验证使用 installFromAIRFile() 方法发送的文件后调度此事件。此事件具有以下属性：
  - available — 如果存在不同于当前应用程序版本的可用版本，则设置为 true ；否则 （版本相同），设置为 false。
  - version — 表示新的可用版本的字符串。
  - path — 表示更新文件的本机路径。

  如果 StatusFileUpdateEvent 对象的可用属性设置为 True，则可以取消此事件。取消该事件可阻止更新继续进行。调用 installUpdate() 方法可继续进行取消的更新。
- StatusFileUpdateErrorEvent.FILE_UPDATE_ERROR — 存在错误且 updater 无法安装 AIR 应用程序时调度此事件。

## 取消更新过程

可以调用 cancelUpdate() 方法来取消更新过程：

```
appUpdater.cancelUpdate();
```

此方法可取消所有未处理的下载，删除所有不完整的下载文件，并可重新启动定期检查计时器。

如果 updater 对象正在初始化，则该方法不执行任何操作。

## 本地化 ApplicationUpdaterUI 界面

ApplicationUpdaterUI 类为更新过程提供默认的用户界面。该用户界面包括允许用户启动过程、取消过程以及执行其他相关操作的对话框。

使用更新描述符文件的 description 元素，您可以用多种语言定义应用程序的描述。使用定义 lang 属性的多个 text 元素，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
     <update xmlns="http://ns.adobe.com/air/framework/update/description/1.0">
       <version>1.1a1</version>
       <url>http://example.com/updates/sample_1.1a1.air</url>
       <description>
           <text xml:lang="en">English description</text>
           <text xml:lang="fr">French description</text>
           <text xml:lang="ro">Romanian description</text>
       </description>
    </update>
```

更新框架将使用最适合最终用户的本地化链的描述。有关详细信息，请参阅 " 定义更新描述符文件并将 AIR 文件添加到 Web 服务器 "。

Flex 开发人员可以直接将新的语言添加到 "ApplicationUpdaterDialogs" 包中。

JavaScript 开发人员可以调用 updater 对象的 addResources() 方法。此方法可动态地添加某种语言的新资源包。该资源包定义某种语言的本地化字符串。这些字符串用于各种对话框的文本字段。

JavaScript 开发人员可以使用 ApplicationUpdaterUI 类的 localeChain 属性来定义用户界面所使用的区域设置链。通常只有 JavaScript (HTML) 开发人员使用此属性。 Flex 开发人员可以使用 ResourceManager 管理区域设置链。

例如，以下 JavaScript 代码定义了罗马尼亚语和匈牙利语的资源包：

```
appUpdater.addResources("ro_RO",
                 {titleCheck: "Titlu", msgCheck: "Mesaj", btnCheck: "Buton"});
appUpdater.addResources("hu", {titleCheck: "Cím", msgCheck: "Üzenet"});
var languages = ["ro", "hu"];
languages = languages.concat(air.Capabilities.languages);
var sortedLanguages = air.Localizer.sortLanguagesByPreference(languages,
                     air.Capabilities.language,
                     "en-US");
sortedLanguages.push("en-US");
appUpdater.localeChain = sortedLanguages;
```

有关详细信息，请参阅语言参考中对 ApplicationUpdaterUI 类的 addResources() 方法的说明。

# 第 18 章：查看源代码

就像用户可以在 Web 浏览器中查看 HTML 页面的源代码一样，用户也可以查看基于 HTML 的 AIR 应用程序的源代码。Adobe® AIR® SDK 包含 AIRSourceViewer.js JavaScript 文件，您可以在应用程序中使用该文件，以便轻松地向最终用户显示源代码。

## 加载、配置和打开 Source Viewer

Source Viewer 代码包含在 JavaScript 文件 AIRSourceViewer.js 中，该文件包含在 AIR SDK 的 frameworks 目录中。若要在应用程序中使用 Source Viewer，请将 AIRSourceViewer.js 复制到应用程序的项目目录中，并在应用程序的 HTML 主文件中通过脚本标签加载该文件：

```
<script type="text/javascript" src="AIRSourceViewer.js"></script>
```

AIRSourceViewer.js 文件定义一个 SourceViewer 类，您可以通过从 JavaScript 代码调用 air.SourceViewer 来访问该类。

SourceViewer 类定义三种方法：getDefault()、 setup() 和 viewSource()。

| 方法 | 说明 |
| --- | --- |
| getDefault() | 静态方法。返回 SourceViewer 实例，可用于调用其他方法。 |
| setup() | 对 Source Viewer 应用配置设置。有关详细信息，请参阅第 214 页的 “配置 Source Viewer” |
| viewSource() | 打开一个新窗口，用户可以在其中浏览和打开主机应用程序的源文件。 |

注：使用 Source Viewer 的代码必须位于应用程序安全沙箱 （在应用程序目录中的文件中）中。

例如，下面的 JavaScript 代码可以实例化 Source Viewer 对象和打开列出所有源文件的 Source Viewer 窗口：

```
var viewer = air.SourceViewer.getDefault();
viewer.viewSource();
```

### 配置 Source Viewer

config() 方法对 Source Viewer 应用给定设置。此方法使用一个参数：configObject。 configObject 对象具有用于为 Source Viewer 定义配置设置的属性。这些属性为 default、 exclude、 initialPosition、 modal、 typesToRemove 和 typesToAdd。

#### default

指定要在 Source Viewer 中显示的初始文件相对路径的字符串。

例如，下面的 JavaScript 代码可以打开 Source Viewer 窗口，其中将 index.html 文件作为初始文件显示：

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.default = "index.html";
viewer.viewSource(configObj);
```

#### exclude

指定要从 Source Viewer 列表中排除的文件或目录的字符串数组。其路径相对于应用程序目录。不支持通配符。

例如，下面的 JavaScript 代码可以打开 Source Viewer 窗口，该窗口列出了除 AIRSourceViewer.js 文件以外的所有源文件以及 Images 和 Sounds 子目录中的文件：

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.exclude = ["AIRSourceViewer.js", "Images" "Sounds"];
viewer.viewSource(configObj);
```

### initialPosition

包含两个数字的数组，用于指定 Source Viewer 窗口的初始 x 坐标和 y 坐标。

例如，下面的 JavaScript 代码可以在屏幕坐标 [40, 60]（X = 40， Y = 60）位置打开 Source Viewer 窗口：

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.initialPosition = [40, 60];
viewer.viewSource(configObj);
```

### modal

一个布尔值，用于指定 Source Viewer 应为模式 (true) 窗口还是非模式 (false) 窗口。默认情况下，Source Viewer 窗口为模式窗口。

例如，下列 JavaScript 代码将打开 Source Viewer 窗口，从而用户可以与 Source Viewer 窗口及任何应用程序窗口进行交互：

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.modal = false;
viewer.viewSource(configObj);
```

### typesToAdd

指定除所包含的默认类型之外，要在 Source Viewer 列表中包含的文件类型的字符串数组。

默认情况下， Source Viewer 将列出以下文件类型：

- 文本文件 - TXT、 XML、 MXML、 HTM、 HTML、 JS、 AS、 CSS、 INI、 BAT、 PROPERTIES、 CONFIG
- 图像文件 - JPG、 JPEG、 PNG、 GIF

  如果未指定任何值，则将包含所有默认类型 （那些在 typesToExclude 属性中指定的类型除外）。

  例如，下面的 JavaScript 代码可以打开包含 VCF 文件和 VCARD 文件的 Source Viewer 窗口：

  ```
  var viewer = air.SourceViewer.getDefault();
  var configObj = {};
  configObj.typesToAdd = ["text.vcf", "text.vcard"];
  viewer.viewSource(configObj);
  ```

  对于列出的每个文件类型，必须指定 “text” （对于文本文件类型）或 “image” （对于图像文件类型）。

### typesToExclude

指定要从 Source Viewer 中排除的文件类型的字符串数组。

默认情况下， Source Viewer 将列出以下文件类型：

- 文本文件 - TXT、 XML、 MXML、 HTM、 HTML、 JS、 AS、 CSS、 INI、 BAT、 PROPERTIES、 CONFIG
- 图像文件 - JPG、 JPEG、 PNG、 GIF

  例如，下面的 JavaScript 代码可以打开未列出 GIF 文件或 XML 文件的 Source Viewer 窗口：

```
var viewer = air.SourceViewer.getDefault();
var configObj = {};
configObj.typesToExclude = ["image.gif", "text.xml"];
viewer.viewSource(configObj);
```

对于列出的每个文件类型，必须指定 “text” （对于文本文件类型）或 “image” （对于图像文件类型）。

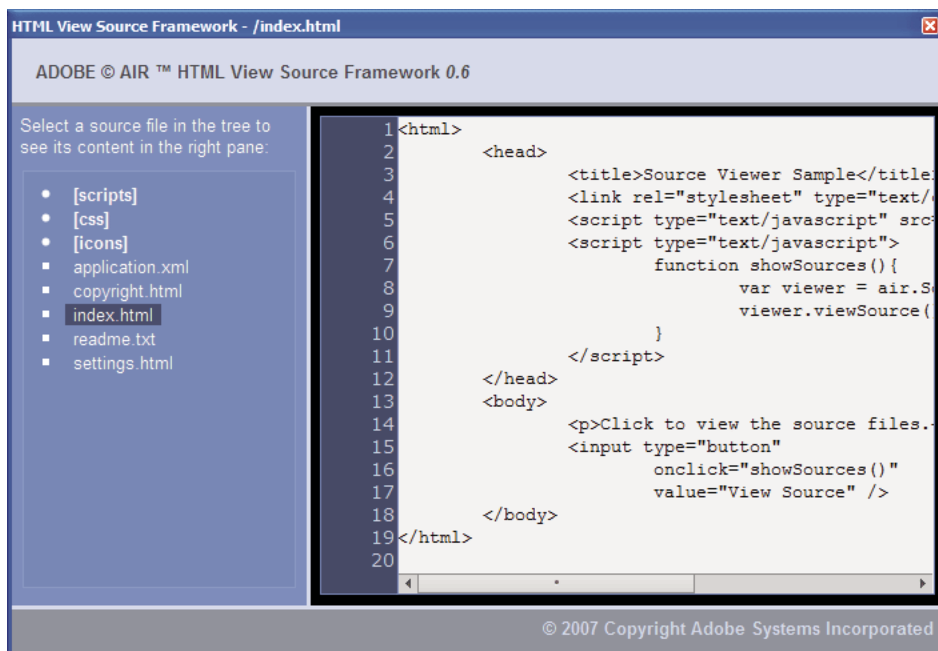
## 打开 Source Viewer

应该包含用户界面元素 （如，链接、按钮或菜单命令），用于在用户选择它时调用 Source Viewer 代码。例如，以下简单的应用程序将在用户单击某个链接时 Source Viewer：

```
<html>
    <head>
        <title>Source Viewer Sample</title>
        <script type="text/javascript" src="AIRSourceViewer.js"></script>
        <script type="text/javascript">
            function showSources(){
                var viewer = air.SourceViewer.getDefault();
                viewer.viewSource()
            }
        </script>
    </head>
    <body>
        <p>Click to view the source files.</p>
        <input type="button"
            onclick="showSources()"
            value="View Source" />
    </body>
</html>
```

# Source Viewer 用户界面

当应用程序调用 SourceViewer 对象的 viewSource() 方法时， AIR 应用程序将打开 Source Viewer 窗口。该窗口包含一个源文件和目录的列表 （在左侧），以及显示所选文件的源代码的显示区域 （在右侧）：




目录以带括号的形式列出。用户可以单击大括号来展开或折叠目录列表。

Source Viewer 可以显示带有可识别扩展名（如 HTML、 JS、 TXT、 XML 及其他）的文本文件的源代码，也可以显示带有可识别图像扩展名（JPG、JPEG、PNG 和 GIF）的图像文件的源代码。如果用户选择不带可识别文件扩展名的文件，则会显示错误消息 （“ 无法从此文件类型获取文本内容 ”(Cannot retrieve text content from this filetype)）。

不会列出通过 setup() 方法排除的任何源文件 （请参阅第 214 页的 “ 加载、配置和打开 Source Viewer”）。

# 第 19 章 : 使用 AIR HTML 内部检查器进行调试

Adobe® AIR® SDK 包含 AIRIntrospector.js JavaScript 文件，您可以应用程序中包含该文件，协助调试基于 HTML 的应用程序。

## 关于 AIR 内部检查器

Adobe AIR HTML/JavaScript 应用程序内部检查器 （称为 AIR HTML 内部检查器）提供了一些有用的功能，可有助于进行基于 HTML 的应用程序的开发和调试工作：

- 它包含一个内部检查器工具，借助此工具，您可以指向应用程序中的用户界面元素，并且可以查看元素的标记和 DOM 属性。
- 它包含用来发送对象引用以进行内部检查的控制台，并且您可以调整属性值和执行 JavaScript 代码。您还可以针对控制台为对象进行序列化，这将对数据编辑加以限制。您还可以从该控制台复制并保存文本。
- 它包含 DOM 属性和函数的树视图。
- 可用于编辑 DOM 元素的属性和文本节点。
- 它列出了应用程序中加载的链接、 CSS 样式、图像和 JavaScript 文件。
- 可用于查看用户界面的初始 HTML 源代码和当前标记源代码。
- 可用于访问应用程序目录中的文件。（此功能仅当为应用程序沙箱打开 AIR HTML 内部检查器控制台时才可用。当为非应用程序沙箱内容打开控制台时不可用。）
- 它包含 XMLHttpRequest 对象及其属性 （包括 responseText 和 responseXML 属性 （如果可用））的查看器。
- 您可以在源代码和文件中搜索匹配文本。

## 加载 AIR 内部检查器代码

AIR 内部检查器代码包含在 AIRIntrospector.js JavaScript 文件中，该文件包含在 AIR SDK 的框架目录中。若要在应用程序中使用 AIR 内部检查器，请将 AIRIntrospector.js 复制到应用程序的项目目录中，并在应用程序的 HTML 主文件中通过脚本标签加载该文件：

```
<script type="text/javascript" src="AIRIntrospector.js"></script>
```

此外，还要在与应用程序中不同本机窗口相对应的每个 HTML 文件中包含该文件。

重要说明：仅当在开发和调试应用程序时才包含 AIRIntrospector.js 文件。在分发的打包 AIR 应用程序中删除该文件。

AIRIntrospector.js 文件定义一个 Console 类，您可以通过从 JavaScript 代码调用 air.Introspector.Console 来访问该类。

注：使用 AIR 内部检查器的代码必须位于应用程序安全沙箱 （在应用程序目录的文件中）中。

## 在控制台选项卡中检查对象

Console 类定义五种方法：log()、 warn()、 info()、 error() 和 dump()。

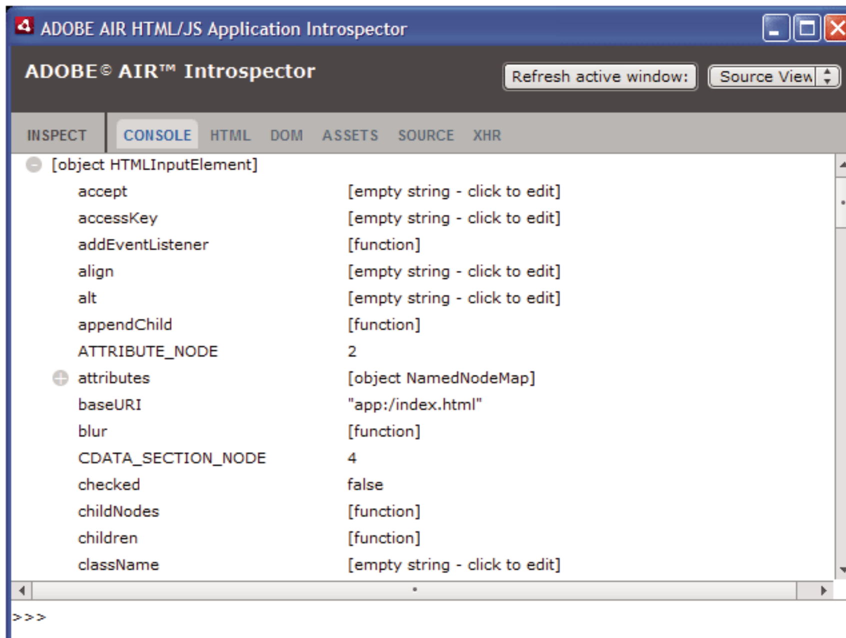
log()、 warn()、info() 和 error() 方法均可用于向 “ 控制台 ” 选项卡发送对象。这些方法中最基本的方法是 log() 方法。以下代码将 test 变量表示的简单对象发送给 “ 控制台 ” 选项卡：

```
var test = "hello";
air.Introspector.Console.log(test);
```

不过，在向 “ 控制台 ” 选项卡发送复杂对象时，它更为有用。例如，以下 HTML 页面包含按钮 (btn1)，此按钮调用向 “ 控制台 ” 选项卡发送按钮对象本身的函数：

```
<html>
    <head>
        <title>Source Viewer Sample</title>
        <script type="text/javascript" src="scripts/AIRIntrospector.js"></script>
        <script type="text/javascript">
            function logBtn()
            {
                var button1 = document.getElementById("btn1");
                air.Introspector.Console.log(button1);
            }
        </script>
    </head>
    <body>
        <p>Click to view the button object in the Console.</p>
        <input type="button" id="btn1"
            onclick="logBtn()"
            value="Log" />
    </body>
</html>
```

当您单击该按钮时， “ 控制台 ” 选项卡会显示 btn1 对象，然后您可以展开该对象的树视图以检查其属性：




您可以通过单击属性名称右侧的列表并修改文本列表来编辑对象的属性。

info()、 error() 和 warn() 方法与 log() 方法类似。不过，当您调用这些方法时，控制台会在行的开头显示图标：

| 方法 | 图标 |
| --- | --- |
| info() | |
| error() | |
| warn() | |

log()、 warn()、 info() 和 error() 方法仅发送实际对象的引用，因此可用属性是那些查看时的属性。如果要对实际对象进行序列化，请使用 dump() 方法。该方法具有两个参数：

| 参数 | 说明 |
| --- | --- |
| dumpObject | 要进行序列化处理的对象。 |
| levels | 要在对象树中检查的最大级别数 （除根级别之外）。默认值为 1 （表示显示树的除根级别之外的一个级别）。此参数是可选的。 |

调用 dump() 方法会在将对象发送至 " 控制台 " 选项卡之前对其进行序列化，这样您就无法编辑对象属性。例如，请看以下代码：

```
var testObject = new Object();
testObject.foo = "foo";
testObject.bar = 234;
air.Introspector.Console.dump(testObject);
```

当执行此代码时，控制台会显示 testObject 对象及其属性，但您无法在控制台中编辑相应属性值。

# 配置 AIR 内部检查器

您可以通过设置 AIRIntrospectorConfig 全局变量的属性来配置控制台。例如，以下 JavaScript 代码将 AIR 内部检查器配置为按照 100 个字符换行：

```
var AIRIntrospectorConfig = new Object();
AIRIntrospectorConfig.wrapColumns = 100;
```

请务必先设置 AIRIntrospectorConfig 变量的属性，然后再加载 AIRIntrospector.js 文件 （通过 script 标签）。

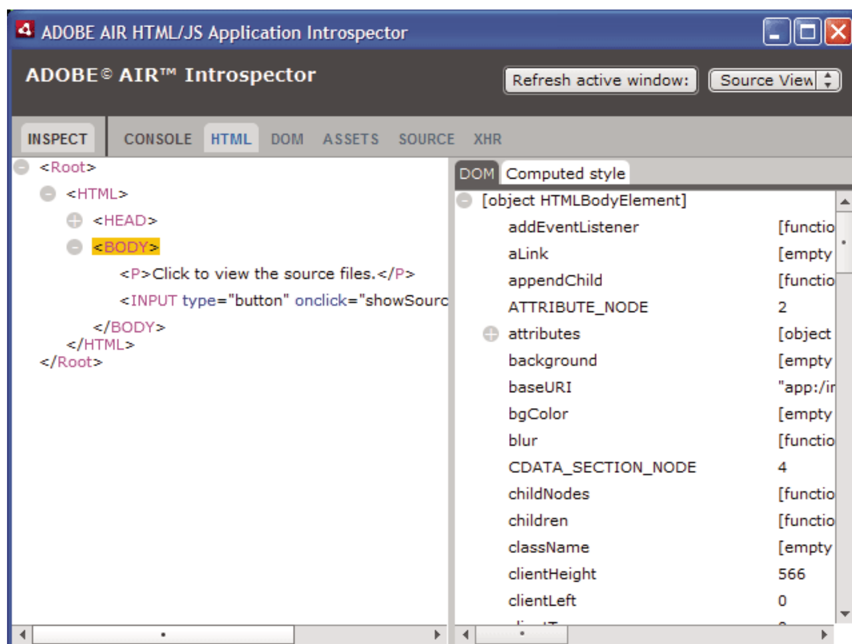
AIRIntrospectorConfig 变量具有八个属性：

| 属性 | 默认值 | 说明 |
| --- | --- | --- |
| closeIntrospectorOnExit | true | 将检查器窗口设置为在应用程序的所有其他窗口都关闭后关闭。 |
| debuggerKey | 123 （F12 键） | 用于显示和隐藏 AIR 内部检查器窗口的键盘快捷键的键控代码。 |
| debugRuntimeObjects | true | 将内部检查器设置为除了展开在 JavaScript 中定义的对象之外，还展开运行时对象。 |
| flashTabLabels | true | 将 " 控制台 " 选项卡和 XMLHttpRequest 选项卡设置为闪烁，以在它们的内容发生变化时 （例如，当在这些选项卡中记录文本时）进行指示。 |
| introspectorKey | 122 （F11 键） | 用于打开 " 检查 "(Inspect) 面板的键盘快捷键的键控代码。 |
| showTimestamp | true | 将 " 控制台 " 选项卡设置为在每行的开头显示时间戳。 |
| showSender | true | 将 " 控制台 " 选项卡设置为在每行的开头显示有关发送消息的对象的信息。 |
| wrapColumns | 2000 | 对源文件换行时的列数。 |

# AIR 内部检查器界面

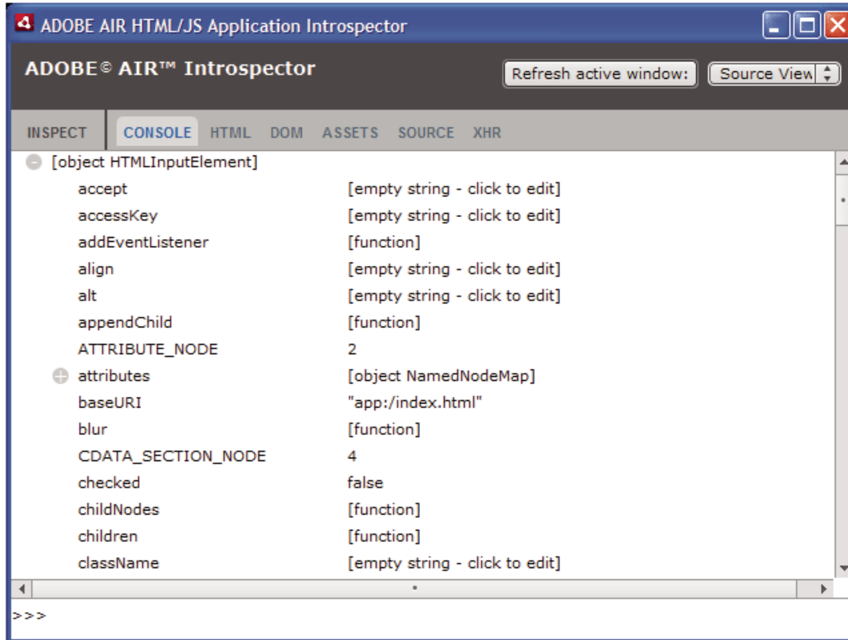
在调试应用程序时，若要打开 AIR 内部检查器窗口，请按 F12 键或调用 Console 类的方法之一（请参阅第 218 页的 “ 在控制台选项卡中检查对象 ”）。您可以将热键配置为 F12 键之外的键；请参阅第 220 页的 “ 配置 AIR 内部检查器 ”。

AIR 内部检查器窗口具有六个选项卡：“ 控制台 ” 选项卡、HTML 选项卡、DOM 选项卡、“ 资源 ” 选项卡、“ 源 ” 选项卡和 XHR 选项卡，如下图所示：



控制台选项卡

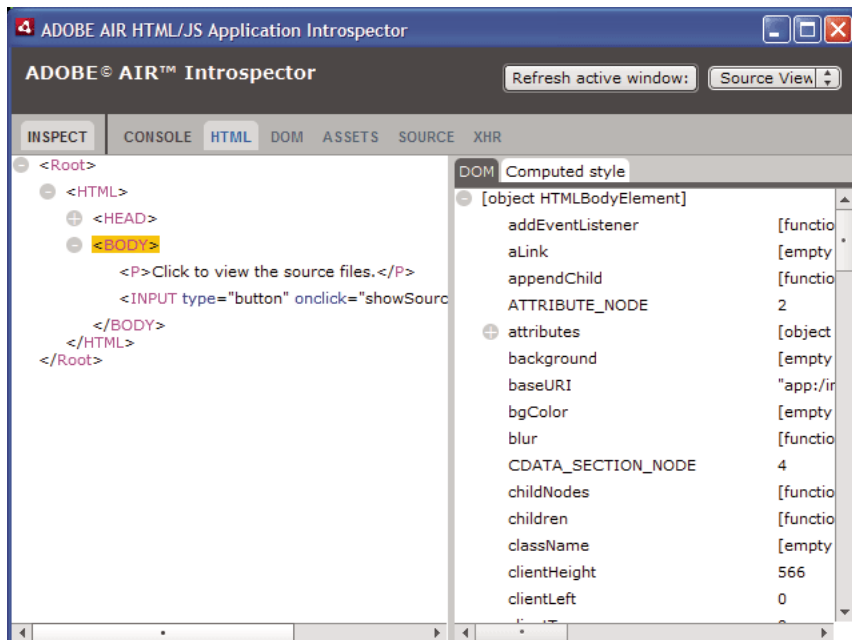
“ 控制台 ” 选项卡显示以参数形式传递给 air.Introspector.Console 类方法之一的属性的值。有关详细信息，请参阅第 218 页的 “ 在控制台选项卡中检查对象 ”。




- 若要清除控制台，请右键单击文本并选择 “ 清除控制台 ”(Clear Console)。
- 若要将“控制台”选项卡中的文本保存到文件，请右键单击“控制台”选项卡并选择“将控制台保存到文件”(Save Console To File)。
- 若要将“控制台”选项卡中的文本保存到剪贴板，请右键单击“控制台”选项卡并选择“将控制台保存到剪贴板”(Save Console To Clipboard)。若仅将选定的文本复制到剪贴板，请右键单击相应文本并选择 “ 复制 ”。
- 若要将 Console 类中的文本保存到文件，请右键单击 “ 控制台 ” 选项卡并选择 “ 将控制台保存到文件 ”(Save Console To File)。
- 若要搜索该选项卡中显示的匹配文本，请单击 Ctrl+F (Windows) 或 Command+F (Mac OS)。（仅搜索可见树节点。）

**HTML** 选项卡

HTML 选项卡可用于以树结构查看整个 HTML DOM。单击某个元素可在该选项卡的右侧查看其属性。单击 + 和 - 图标可展开和折叠树中的节点。




您可以在 HTML 选项卡中编辑任何属性或文本元素，编辑的值会在应用程序中反映出来。

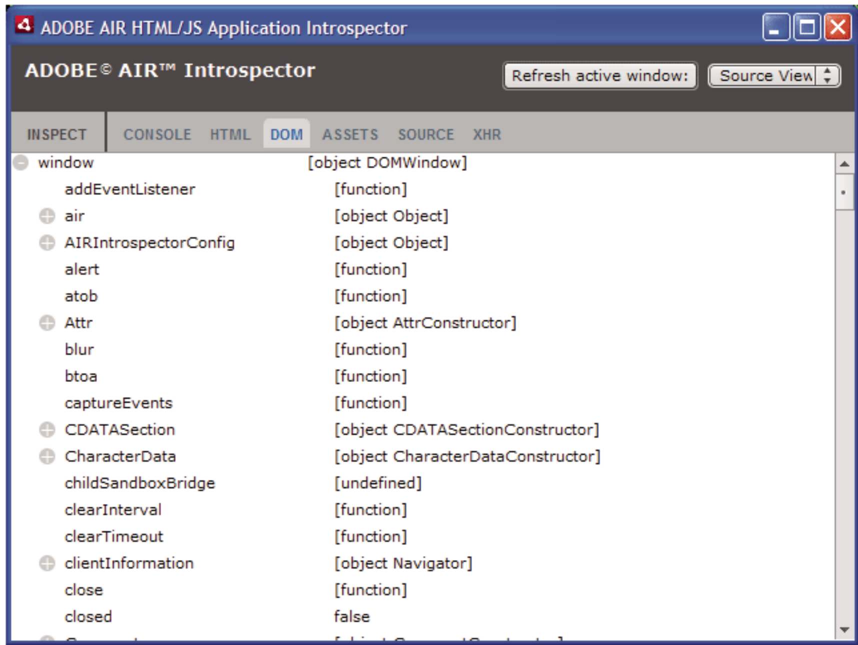
单击 " 检查 "(Inspect) 按钮 （在 AIR 内部检查器窗口中选项卡列表的左侧）。您可以单击主窗口的 HTML 页面中的任何元素，关联的 DOM 对象会显示在 HTML 选项卡中。如果主窗口具有焦点，则您还可以通过按键盘快捷键打开或关闭 " 检查 "(Inspect) 按钮。默认情况下， F11 为键盘快捷键。您可以将键盘快捷键配置为 F11 键之外的键；请参阅第 220 页的 " 配置 AIR 内部检查器 "。

单击 " 刷新活动窗口 "(Refresh Active Window) 按钮 （在 AIR 内部检查器窗口的顶部）可刷新 HTML 选项卡中显示的数据。

单击 Ctrl+F (Windows) 或 Command+F (Mac OS) 可搜索该选项卡中显示的匹配文本。（仅搜索可见树节点。）

**DOM** 选项卡

DOM 选项卡以树结构显示窗口对象。您可以编辑任何字符串和数值属性，编辑的值会在应用程序中反映出来。

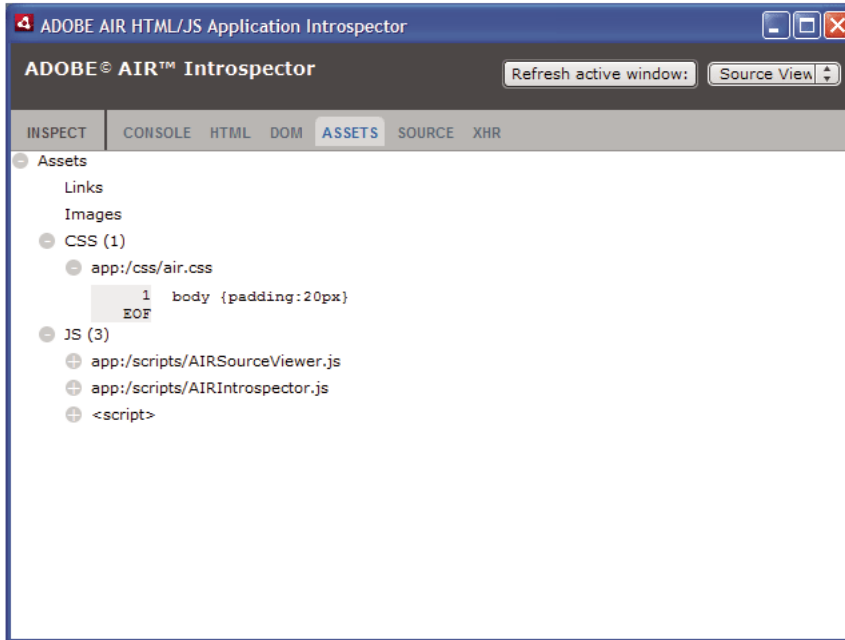



单击 " 刷新活动窗口 "(Refresh Active Window) 按钮 （在 AIR 内部检查器窗口的顶部）可刷新 DOM 选项卡中显示的数据。

单击 Ctrl+F (Windows) 或 Command+F (Mac OS) 可搜索该选项卡中显示的匹配文本。（仅搜索可见树节点。）

资源选项卡

“ 资源 ” 选项卡可用于检查本机窗口中加载的链接、图像、 CSS 和 JavaScript 文件。展开这些节点之一将显示文件的内容或显示所使用的实际图像。




单击 “ 刷新活动窗口 ”(Refresh Active Window) 按钮 （在 AIR 内部检查器窗口的顶部）可刷新 “ 资源 ” 选项卡中显示的数据。

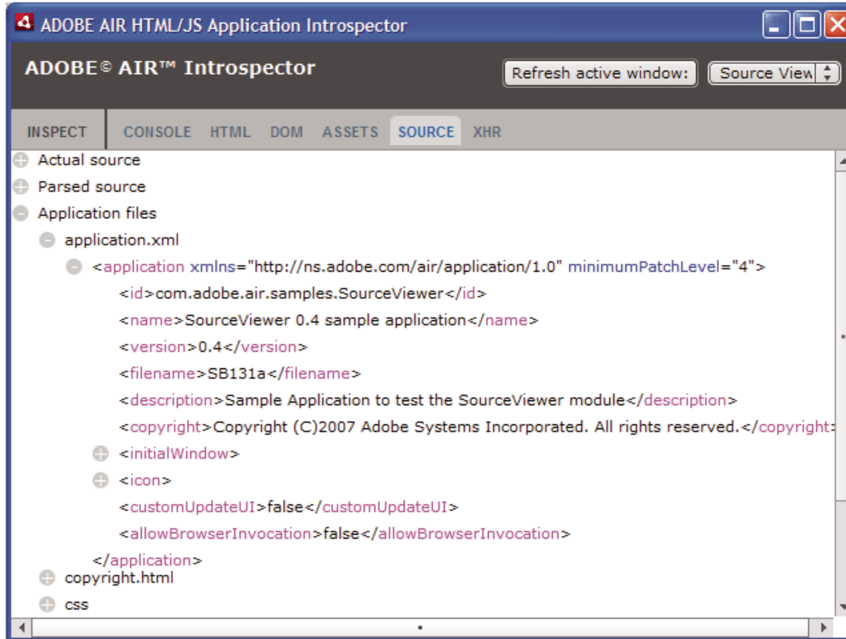
单击 Ctrl+F (Windows) 或 Command+F (Mac OS) 可搜索该选项卡中显示的匹配文本。（仅搜索可见树节点。）

源选项卡

“ 源 ” 选项卡包含以下三个部分：

- 实际源代码 - 显示应用程序启动时作为根内容加载的页面的 HTML 源代码。
- 已解析源代码 - 显示构成应用程序 UI 的当前标记，它可能与实际源代码不同，因为应用程序采用 Ajax 技术动态生成标记代码。

- 应用程序文件 - 列出应用程序目录中的文件。此列表仅当从应用程序安全沙箱中的内容启动 AIR 内部检查器时才可供 AIR 内部检查器使用。在本部分中，您既可以查看文本文件的内容，也可以查看图像。

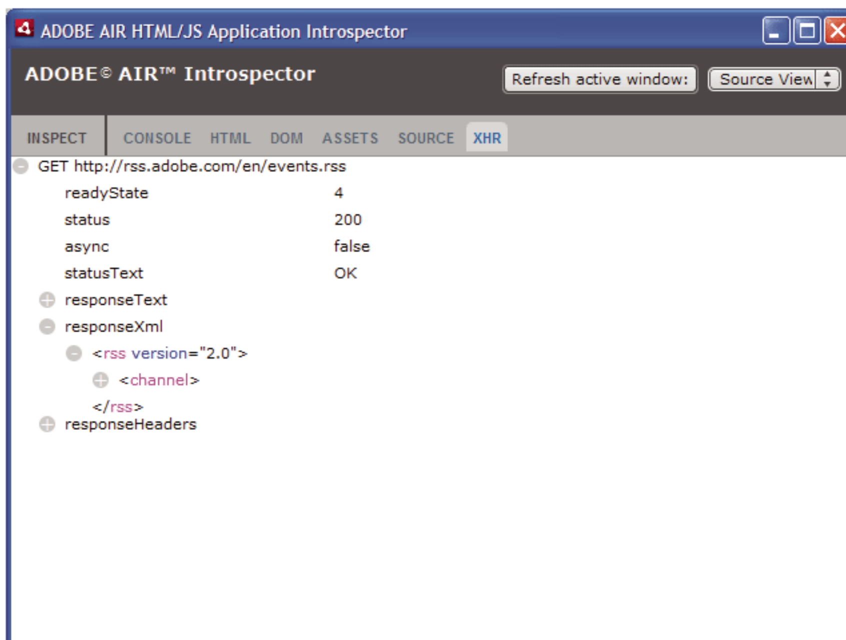



单击 " 刷新活动窗口 "(Refresh Active Window) 按钮（在 AIR 内部检查器窗口的顶部）可刷新 " 源 " 选项卡中显示的数据。

单击 Ctrl+F (Windows) 或 Command+F (Mac OS) 可搜索该选项卡中显示的匹配文本。（仅搜索可见树节点。）

### XHR 选项卡

XHR 选项卡截获应用程序中的所有 XMLHttpRequest 通信并记录相关信息。这样，您即可在树视图中查看 XMLHttpRequest 属性（包括 responseText 和 responseXML（如果可用））。



单击 Ctrl+F (Windows) 或 Command+F (Mac OS) 可搜索该选项卡中显示的匹配文本。（仅搜索可见树节点。）

# 对非应用程序沙箱中的内容使用 AIR 内部检查器

可以将内容从应用程序目录加载到映射到非应用程序沙箱的 iframe 或 frame。（请参阅 Adobe AIR 中的 HTML 安全性（针对 ActionScript 开发人员）或 Adobe AIR 中的 HTML 安全性（针对 HTML 开发人员））。您可以对此类内容使用 AIR 内部检查器，但要遵循以下规则：

- AIRIntrospector.js 文件必须同时包含在应用程序沙箱和非应用程序沙箱 (iframe) 内容中。
- 请勿覆盖 parentSandboxBridge 属性； AIR 内部检查器代码将使用此属性。根据需要添加属性。因此，请不要按如下方式编写代码：

  ```
  parentSandboxBridge = mytrace: function(str) {runtime.trace(str)}} ;
  ```

  使用如下语法：

  ```
  parentSandboxBridge.mytrace = function(str) {runtime.trace(str)};
  ```

- 从非应用程序沙箱内容，您无法通过按 F12 键或调用 air.Introspector.Console 类中的方法之一打开 AIR 内部检查器。您只能通过单击 " 打开内部检查器 "(Open Introspector) 按钮来打开内部检查器窗口。默认情况下，该按钮添加在 iframe 或帧的右上角。（由于对非应用程序沙箱内容施加安全限制，因此只能通过用户执行一定的手势来打开新窗口，如单击按钮。）
- 您可以为应用程序沙箱和非应用程序沙箱打开单独的 AIR 内部检查器窗口。您可以通过 AIR 内部检查器窗口中显示的标题来区分二者。
- 当从非应用程序沙箱运行 AIR 内部检查器时， " 源 " 选项卡不显示应用程序文件。
- AIR 内部检查器只能查看从其打开它的沙箱中的代码。

# 第 20 章 : 本地化 AIR 应用程序

**Adobe AIR 1.1** 和更高版本

Adobe® AIR® 中包括对多种语言的支持。

有关本地化 ActionScript 3.0 和 Flex 框架中内容的概述，请参阅 《ActionScript 3.0 开发人员指南》中的 " 本地化应用程序 "。

**AIR** 中支持的语言

在 AIR 1.1 发行版中引入了对下列语言的 AIR 应用程序的本地化支持：

- 简体中文
- 繁体中文
- 法语
- 德语
- 意大利语
- 日语
- 朝鲜语
- 巴西葡萄牙语
- 俄语
- 西班牙语

在 AIR 1.5 发行版中，添加了下列语言：

- 捷克语
- 荷兰语
- 波兰语
- 瑞典语
- 土耳其语

**更多帮助主题**

在 Adobe AIR 中构建多语言 Flex 应用程序

构建基于 HTML 的多语言应用程序

## 本地化 AIR 应用程序安装程序中的应用程序名称和说明

**Adobe AIR 1.1** 和更高版本

您可以为应用程序描述符文件中的 name 和 description 元素指定多种语言。例如，以下示例用三种语言 （英语、法语和德语）指定应用程序名称：

```
<name>
    <text xml:lang="en">Sample 1.0</text>
    <text xml:lang="fr">Échantillon 1.0</text>
    <text xml:lang="de">Stichprobe 1.0</text>
</name>
```

每个文本元素的 xml:lang 属性用于指定语言代码，有关具体定义，请参阅 RFC4646 (http://www.ietf.org/rfc/rfc4646.txt)。

name 元素定义了 AIR 应用程序安装程序显示的应用程序名称。AIR 应用程序安装程序使用与操作系统设置定义的用户界面语言最匹配的本地化值。

同样，也可以在应用程序描述符文件中指定 description 元素的多种语言版本。该元素定义了 AIR 应用程序安装程序显示的说明文本。

这些设置仅适用于可在 AIR 应用程序安装程序中使用的语言。这些设置不定义可供运行的已安装应用程序使用的区域设置。AIR 应用程序提供的用户界面可支持多种语言，包括但不限于 AIR 应用程序安装程序可用的语言。

有关详细信息，请参阅第 157 页的 “AIR 应用程序描述符元素 ”。

**更多帮助主题**

在 Adobe AIR 中构建多语言 Flex 应用程序

构建基于 HTML 的多语言应用程序

# 使用 AIR HTML 本地化框架本地化 HTML 内容

**Adobe AIR 1.1** 和更高版本

AIR 1.1 SDK 包括一个 HTML 本地化框架。AIRLocalizer.js JavaScript 文件定义了该框架。AIR SDK 的 frameworks 目录包含 AIRLocalizer.js 文件。该文件包含一个 air.Localizer 类，该类提供了帮助创建可支持多个本地化版本的应用程序的功能。

## 加载 AIR HTML 本地化框架代码

若要使用本地化框架，请将 AIRLocalizer.js 文件复制到您的项目中。然后使用 script 标签将其包括在应用程序的主要 HTML 文件中：

```
<script src="AIRLocalizer.js" type="text/javascript" charset="utf-8"></script>
```

后续的 JavaScript 可以调用 air.Localizer.localizer 对象：

```
<script>
    var localizer = air.Localizer.localizer;
</script>
```

air.Localizer.localizer 对象是定义使用和管理本地化资源的方法和属性的单个对象。 Localizer 类包含以下方法：

| 方法 | 说明 |
|---|---|
| getFile() | 获取指定区域设置的指定资源包。请参阅第 234 页的 “ 获取特定区域设置的资源 ”。 |
| getLocaleChain() | 返回区域设置链中的语言。请参阅第 233 页的 “ 定义区域设置链 ”。 |
| getResourceBundle() | 将捆绑密钥和相应的值作为对象返回。请参阅第 234 页的 “ 获取特定区域设置的资源 ”。 |
| getString() | 获取为资源定义的字符串。请参阅第 234 页的 “ 获取特定区域设置的资源 ”。 |
| setBundlesDirectory() | 设置包目录的位置。请参阅第 233 页的 “ 自定义 AIR HTML Localizer 设置 ”。 |

| 方法 | 说明 |
|---|---|
| setLocalAttributePrefix() | 设置由用于 HTML DOM 元素中的 localizer 属性使用的前缀。请参阅第 233 页的 " 自定义 AIR HTML Localizer 设置 "。 |
| setLocaleChain() | 设置区域设置链中语言的顺序。请参阅第 233 页的 " 定义区域设置链 "。 |
| sortLanguagesByPreference() | 基于操作系统设置中区域设置的顺序，对区域设置链中的区域设置进行排序。请参阅第 233 页的 " 定义区域设置链 "。 |
| update() | 使用当前区域设置链中的本地化字符串更新 HTML DOM （或 DOM 元素）。有关区域设置链的说明，请参阅第 231 页的 " 管理区域设置链 "。有关 update() 方法的详细信息，请参阅第 232 页的 " 更新 DOM 元素以使用当前区域设置 "。 |

Localizer 类包含以下静态属性：

| 属性 | 说明 |
|---|---|
| localizer | 返回对应用程序单个 Localizer 对象的引用。 |
| ultimateFallbackLocale | 应用程序不支持用户首选参数时使用的区域设置。请参阅第 233 页的 " 定义区域设置链 "。 |

## 定义资源包

HTML 本地化框架从本地化 文件中读取字符串的本地化版本。本地化文件是文本文件中经过序列化的基于键的值集合。本地化文件有时被称为包。

创建一个名为 locale 的应用程序项目目录的子目录。（您也可以使用其他名称，请参阅第 233 页的 " 自定义 AIR HTML Localizer 设置 "。）该目录将包括本地化文件。该目录被称为包目录。

针对应用程序支持的每个区域设置，都创建一个包目录的子目录。命名每个子目录以与区域设置代码相匹配。例如，将法语目录命名为 "fr" 并将英语目录命名为 "en"。您可以使用下划线 (_) 字符定义具有语言和国家 / 地区代码的区域设置。例如，将美式英语目录命名为 "en_us"。（也可以使用连字符代替下划线，如 "en-us"。 HTML 本地化框架支持这两种形式。）

您可以向 locale 子目录添加任意数目的资源文件。通常，为每种语言均创建一个本地化文件 （并将该文件放在该语言的目录中）。 HTML 本地化框架包括 getFile() 方法，利用该方法可以读取文件的内容 （请参阅第 234 页的 " 获取特定区域设置的资源 "。

具有 .properties 文件扩展名的文件被认为是本地化属性文件。可以使用这些文件为区域设置定义键值对。属性文件在每行都定义了一个字符串值。例如，以下定义了一个字符串值 "Hello in English."，针对名为 greeting 的键：

```
greeting=Hello in English.
```

包含以下文本的属性文件定义了六个键值对：

```
title=Sample Application
greeting=Hello in English.
exitMessage=Thank you for using the application.
color1=Red
color2=Green
color3=Blue
```

此示例显示了要存储在 en 目录中的属性文件的英语版本。

此属性文件的法语版本放置在 fr 目录中：

```
title=Application Example
greeting=Bonjour en français.
exitMessage=Merci d'avoir utilisé cette application.
color1=Rouge
color2=Vert
color3=Bleu
```

您可以为不同种类的信息定义多个资源文件。例如， legal.properties 文件可能包含法律文本样本 （如版权信息）。您可以在多个应用程序中重复使用该资源。同样，您可以定义单独的文件为用户界面的不同部分定义本地化内容。

对这些文件使用 UTF-8 编码以支持多种语言。

## 管理区域设置链

当应用程序加载 AIRLocalizer.js 文件时，该文件将检查应用程序中定义的区域设置。这些区域设置对应于包目录的子目录（请参阅第 230 页的 " 定义资源包 "）。此可用区域设置的列表称为区域设置链。 AIRLocalizer.js 文件将根据操作系统设置定义的首选顺序自动对区域设置链进行排序。（Capabilities.languages 属性按首选顺序列出操作系统用户界面语言。）

因此，如果应用程序为 "en"、"en_US" 和 "en_UK" 区域设置定义了资源，则 AIR HTML Localizer 框架将对区域设置链进行相应排序。当应用程序在将 "en" 报告为主区域设置的系统中启动时，区域设置链的排序顺序为 ["en", "en_US", "en_UK"]。此时，应用程序首先在 "en" 包中查找资源，然后在 "en_US" 包中查找。

但是，如果系统将 "en-US" 报告为主区域设置，则排序将使用 ["en_US", "en", en_UK"]。在此情况下，应用程序首先在 "en_US" 包中查找资源，然后在 "en" 包中查找。

默认情况下，应用程序会将区域设置链中的第一个区域设置定义为要使用的默认区域设置。您可以请用户在第一次运行应用程序时选择一个区域设置。然后，您可以选择将该选择存储在首选参数文件中，并在随后的应用程序启动中使用该区域设置。

应用程序可以在区域设置链中的任何区域设置中使用资源字符串。如果特定区域设置未定义资源字符串，则应用程序将为区域设置链中定义的其他区域设置使用下一个匹配的资源字符串。

可以通过调用 Localizer 对象的 setLocaleChain() 方法自定义区域设置链。请参阅第 233 页的 " 定义区域设置链 "。

## 利用本地化的内容更新 DOM 元素

应用程序中的元素可以引用本地化属性文件中的键值。例如，以下示例中的 title 元素指定了 local_innerHTML 属性。本地化框架使用此属性查找本地化值。默认情况下，框架会查找以 "local_" 开头的属性名称。该框架将更新名称与 "local_" 之后的文本相匹配的属性。在本例中，框架会设置 title 元素的 innerHTML 属性。innerHTML 属性 (attribute) 将在默认属性 (property) 文件 (default.properties) 中使用为 mainWindowTitle 键定义的值：

```
<title local_innerHTML="default.mainWindowTitle"/>
```

如果当前区域设置未定义匹配的值，则 localizer 框架将搜索区域设置链的其余部分。该框架将使用区域设置链中定义了值的下一个区域设置。

在以下示例中， p 元素的文本 （innerHTML 属性 (attribute)）使用在默认属性 (property) 文件中定义的 greeting 键的值：

```
<p local_innerHTML="default.greeting" />
```

在以下示例中， input 元素的值属性 (attribute) （和显示文本）使用默认属性 (property) 文件中定义的 btnBlue 键的值：

```
<input type="button" local_value="default.btnBlue" />
```

若要更新 HTML DOM 以使用当前区域设置链中定义的字符串，请调用 Localizer 对象的 update() 方法。调用 update() 方法将使 Localizer 对象分析 DOM 并在其找到本地化 ("local_...") 属性处应用操作：

```
air.Localizer.localizer.update();
```

您可以为属性 （如 "innerHTML"）及其相应的本地化属性 （如 "local_innerHTML"）都定义值。在这种情况下，如果本地化框架在本地化链中找到一个匹配值，则仅覆盖该属性值。例如，以下元素定义了 value 和 local_value 属性：

```
<input type="text" value="Blue" local_value="default.btnBlue"/>
```

也可以只更新特定的 DOM 元素。请参阅下一节第 232 页的 " 更新 DOM 元素以使用当前区域设置 "。

默认情况下， AIR HTML Localizer 使用 "local_" 作为定义元素的本地化设置的属性前缀。例如，默认情况下，local_innerHTML 属性定义了用于元素的 innerHTML 值的包和资源名称。同样，默认情况下，local_value 属性定义了用于元素的 value 属性的包和资源名称。您可以将 Localizer 配置为使用除 "local_" 之外的属性前缀。请参阅第 233 页的 " 自定义 AIR HTML Localizer 设置 "。

## 更新 DOM 元素以使用当前区域设置

当 Localizer 对象更新 HTML DOM 时，将导致已标记元素根据当前区域设置链中定义的字符串使用属性值。若要使 HTML localizer 更新 HTML DOM，请调用 Localizer 对象的 update() 方法：

```
air.Localizer.localizer.update();
```

若要仅更新指定的 DOM 元素，请将其作为参数传递到 update() 方法。update() 方法只有一个参数 parentNode，该参数为可选参数。指定后， parentNode 参数将定义要本地化的 DOM 元素。调用 update() 方法并指定 parentNode 参数，可将为所有指定本地化属性的子元素设置本地化值。

以下面的 div 元素为例：

```
<div id="colorsDiv">
    <h1 local_innerHTML="default.lblColors" ></h1>
    <p><input type="button" local_value="default.btnBlue" /></p>
    <p><input type="button" local_value="default.btnRed" /></p>
    <p><input type="button" local_value="default.btnGreen" /></p>
</div>
```

若要更新此元素以使用当前区域设置链中定义的本地化字符串，请使用以下 JavaScript 代码：

```
var divElement = window.document.getElementById("colorsDiv");
air.Localizer.localizer.update(divElement);
```

如果并未在区域设置链中找到键值，则本地化框架将属性值设置为 "local_" 属性的值。例如，在上一个示例中，假设本地化框架无法找到 lblColors 键 （在区域设置链中的任何一个 default.properties 文件中）的值。在此情况下，它将使用 "default.lblColors" 作为 innerHTML 值。使用该值指示 （开发人员）缺少资源。

当 update() 方法在区域设置链中无法找到资源时，将调度 resourceNotFound 事件。air.Localizer.RESOURCE_NOT_FOUND 常量定义了字符串 "resourceNotFound"。该事件具有三个属性：bundleName、 resourceName 和 locale。 bundleName 属性是在其中未找到该资源的包的名称。 resourceName 属性是在其中未找到该资源的包的名称。 locale 属性是在其中未找到该资源的区域设置的名称。

当 update() 方法无法找到指定包时，将调度 bundleNotFound 事件。 air.Localizer.BUNDLE_NOT_FOUND 常量定义了字符串 "bundleNotFound"。该事件具有两个属性：bundleName 和 locale。bundleName 属性是在其中未找到该资源的包的名称。locale 属性是在其中未找到该资源的区域设置的名称。

update() 方法采用异步方式操作 （并异步调度 resourceNotFound 和 bundleNotFound 事件）。以下代码将为 resourceNotFound 和 bundleNotFound 事件设置事件侦听器：

```
air.Localizer.localizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, rnfHandler);
air.Localizer.localizer.update();
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":." + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":." + event.locale);
}
```

## 自定义 AIR HTML Localizer 设置

利用 Localizer 对象的 setBundlesDirectory() 方法可以自定义包目录路径。利用 Localizer 对象的 setLocalAttributePrefix() 方法可以自定义包目录路径并可自定义 Localizer 使用的属性值。

默认的包目录定义为应用程序目录的区域设置子目录。可以通过调用 Localizer 对象的 setBundlesDirectory() 方法指定另一个目录。此方法将使用一个以字符串形式表示的参数 path （它是所需包目录的路径）。 path 参数的值可以为以下任意值：

- 用于定义与应用程序目录有关的路径的字符串，如 "locales"
- 用于定义使用 app、app-storage 或 file URL 方案（如 "app://languages"）（请勿使用 http URL 方案）的有效 URL 的字符串
- File 对象

有关 URL 和目录路径的信息，请参阅：

- File 对象的路径 （针对 ActionScript 开发人员）
- File 对象的路径 （针对 HTML 开发人员）

例如，下列代码将包目录设置为应用程序存储目录的语言子目录 （而非应用程序目录）：

```
air.Localizer.localizer.setBundlesDirectory("languages");
```

将有效的路径作为 path 参数传递。否则，该方法将引发 BundlePathNotFoundError 异常。该错误将 "BundlePathNotFoundError" 作为其 name 属性，而其 message 属性则指定无效的路径。

默认情况下， AIR HTML Localizer 使用 "local_" 作为定义元素的本地化设置的属性前缀。例如， local_innerHTML 属性定义了用于以下 input 元素的 innerHTML 值的包和资源名称：

```
<p local_innerHTML="default.greeting" />
```

利用 Localizer 对象的 setLocalAttributePrefix() 方法，可以使用除 "local_" 之外的属性前缀。此静态方法将使用一个参数，而该参数是要用作属性前缀的字符串。例如，以下代码将本地化框架设置为使用 “loc_” 作为属性前缀：

```
air.Localizer.localizer.setLocalAttributePrefix("loc_");
```

您可以自定义本地化框架使用的属性前缀。如果默认值 ("local_") 与您的代码使用的另一个属性的名称冲突，则您可能希望自定义该前缀。请确保调用此方法时使用对 HTML 属性有效的字符。（例如，该值不能包含空格字符。）

有关在 HTML 元素中使用本地化属性的详细信息，请参阅第 231 页的 “ 利用本地化的内容更新 DOM 元素 ”。

包目录和属性前缀设置在不同的应用程序会话之间不会保留。如果使用自定义包目录或属性前缀设置，请确保每次启动应用程序时对其进行设置。

## 定义区域设置链

默认情况下，加载 AIRLocalizer.js 代码时，它将设置默认的区域设置链。包目录和操作系统语言设置中可用的区域设置定义了该区域设置链。（有关详细信息，请参阅第 231 页的 “ 管理区域设置链 ”。）

可以通过调用 Localizer 对象的静态 setLocaleChain() 方法修改区域设置链。例如，如果用户为某个特定语言指示一个首选参数，则您最好调用此方法。 setLocaleChain() 方法使用一个参数 chain，该参数为一个区域设置数组，如 ["fr_FR","fr","fr_CA"]。数组中区域设置的顺序设定了框架查找资源的顺序 （在后续操作中）。如果未在链中找到第一个区域设置的资源，它将继续查找其他区域设置的资源。如果 chain 参数丢失、不是数组或为空数组，则此功能将失败并引发 IllegalArgumentsError 异常。

Localizer 对象的 getLocaleChain() 静态方法返回一个列出当前区域设置链中的区域设置的数组。

以下代码将读取当前区域设置链并将两个法语区域设置添加到链头：

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
```

当 setLocaleChain() 方法更新区域设置链时，将调度 "change" 事件。 air.Localizer.LOCALE_CHANGE 常量定义了字符串 "change"。该事件具有一个 localeChain 属性，该属性为新的区域设置链中的区域设置代码数组。以下代码为此事件设置了一个事件侦听器：

```
var currentChain = air.Localizer.localizer.getLocaleChain();
newLocales = ["fr_FR", "fr"];
localizer.addEventListener(air.Localizer.LOCALE_CHANGE, changeHandler);
air.Localizer.localizer.setLocaleChain(newLocales.concat(currentChain));
function changeHandler(event)
{
    alert(event.localeChain);
}
```

静态 air.Localizer.ultimateFallbackLocale 属性表示在应用程序不支持用户首选参数时所用的区域设置。默认值为 "en"。您可以将其设置为另一个区域设置，如以下代码中所示：

```
air.Localizer.ultimateFallbackLocale = "fr";
```

## 获取特定区域设置的资源

Localizer 对象的 getString() 方法将返回为特定区域设置中的资源定义的字符串。调用此方法时，不必指定 locale 值。在此情况下，该方法将查看整个区域设置链并返回提供给定资源名称的第一个区域设置中的字符串。此方法具有以下参数：

| 参数 | 说明 |
| --- | --- |
| bundleName | 包含资源的包。这是不带 .properties 扩展名的属性文件的文件名。（例如，如果此参数设置为 "alerts"，则 Localizer 代码将在名为 alerts.properties 的本地化文件中查找。 |
| resourceName | 资源名称。 |
| templateArgs | 可选。用于替换替换字符串中的编号标签的字符串数组。以调用 templateArgs 参数为 ["Raúl", "4"] 且匹配资源字符串为 "Hello, {0} You have {1} new messages."。在此情况下，此函数将返回 "Hello, Raúl. You have 4 new messages."。若要忽略此设置，请传递 null 值。 |
| locale | 可选。要使用的区域设置代码 （如 "en"、 "en_us" 或 "fr"）。如果提供了一个区域设置但未找到匹配值，则此方法将不会继续在区域设置链中的其他区域设置中搜索值。如果未指定任何区域设置代码，则此函数将返回区域设置链中第一个区域设置中为给定的资源名称提供值的字符串。 |

本地化框架可以更新标记的 HTML DOM 属性。但是，可以通过其他方式使用本地化字符串。例如，可以在某些动态生成的 HTML 中使用字符串或在函数调用中将其作为参数值。例如，以下代码将通过 fr_FR 区域设置的默认属性文件中 error114 资源内定义的字符串调用 alert() 函数：

```
alert(air.Localizer.localizer.getString("default", "error114", null, "fr_FR"));
```

当 getString() 方法在指定包中无法找到资源时，将调度 resourceNotFound 事件。air.Localizer.RESOURCE_NOT_FOUND 常量定义了字符串 "resourceNotFound"。该事件具有三个属性：bundleName、 resourceName 和 locale。 bundleName 属性是在其中未找到该资源的包的名称。 resourceName 属性是在其中未找到该资源的包的名称。 locale 属性是在其中未找到该资源的区域设置的名称。

当 getString() 方法无法找到指定包时，将调度 bundleNotFound 事件。air.Localizer.BUNDLE_NOT_FOUND 常量定义了字符串 "bundleNotFound"。该事件具有两个属性：bundleName 和 locale。bundleName 属性是在其中未找到该资源的包的名称。locale 属性是在其中未找到该资源的区域设置的名称。

getString() 方法采用异步方式操作 （并异步调度 resourceNotFound 和 bundleNotFound 事件）。以下代码将为 resourceNotFound 和 bundleNotFound 事件设置事件侦听器：

```
air.Localizerlocalizer.addEventListener(air.Localizer.RESOURCE_NOT_FOUND, rnfHandler);
air.Localizerlocalizer.addEventListener(air.Localizer.BUNDLE_NOT_FOUND, bnfHandler);
var str = air.Localizer.localizer.getString("default", "error114", null, "fr_FR");
function rnfHandler(event)
{
    alert(event.bundleName + ": " + event.resourceName + ":." + event.locale);
}
function bnfHandler(event)
{
    alert(event.bundleName + ":." + event.locale);
}
```

Localizer 对象的 getResourceBundle() 方法为给定的区域设置返回一个指定的绑定。该方法的返回值是其属性与绑定中的键相匹配的对象。（如果应用程序找不到指定的绑定，则该方法返回 null。）

该方法采用两个参数 — locale 和 bundleName。

| 参数 | 说明 |
| --- | --- |
| locale | 区域设置 （如 "fr"）。 |
| bundleName | 绑定名称。 |

例如，以下代码调用 document.write() 方法加载 fr 区域设置的默认绑定。然后，调用 document.write() 方法写入该绑定中 str1 和 str2 键的值：

```
var aboutWin = window.open();
var bundle = localizer.getResourceBundle("fr", "default");
aboutWin.document.write(bundle.str1);
aboutWin.document.write("<br/>");
aboutWin.document.write(bundle.str2);
aboutWin.document.write("<br/>");
```

getResourceBundle() 方法在找不到指定的绑定时将调度 bundleNotFound 事件。air.Localizer.BUNDLE_NOT_FOUND 常量定义了字符串 "bundleNotFound"。该事件具有两个属性：bundleName 和 locale。 bundleName 属性是在其中未找到该资源的包的名称。 locale 属性是在其中未找到该资源的区域设置的名称。

Localizer 对象的 getFile() 方法将以字符串形式为给定区域设置返回包的内容。包文件将以 UTF-8 文件格式读取。此方法具有以下参数：

| 参数 | 说明 |
| --- | --- |
| resourceFileName | 资源文件的文件名 （例如 "about.html"）。 |
| templateArgs | 可选。用于替换替换字符串中的编号标签的字符串数组。以调用 templateArgs 参数为 ["Raúl", "4"] 且匹配的资源文件包含以下两行的函数为例：<br>`<html>`<br>`<body>Hello, {0}. You have {1} new messages.</body>`<br>`</html>`<br>在此情况下，此函数将返回两行字符串：<br>`<html>`<br>`<body>Hello, Raúl. You have 4 new messages. </body>`<br>`</html>` |
| locale | 要使用的区域设置代码，例如 "en_GB"。如果提供了区域设置但未找到匹配的文件，则此方法将不会继续在区域设置链中的其他区域设置中搜索。如果未指定区域设置代码，则此函数将返回区域设置链中第一个区域设置中的文本，而该区域设置链具有一个与 resourceFileName 相匹配的文件。 |

例如，以下代码使用 fr 区域设置的 about.html 文件的内容调用 document.write() 方法：

```
var aboutWin = window.open();
var aboutHtml = localizer.getFile("about.html", null, "fr");
aboutWin.document.close();
aboutWin.document.write(aboutHtml);
```

当 getFile() 方法在区域设置链中无法找到资源时，将调度 fileNotFound 事件。 air.Localizer.FILE_NOT_FOUND 常量定义了字符串 "resourceNotFound"。getFile() 方法采用异步方式操作（并异步调度 fileNotFound 事件）。该事件具有两个属性：fileName 和 locale。fileName 属性为未找到的文件的名称。locale 属性是在其中未找到该资源的区域设置的名称。以下代码为此事件设置了一个事件侦听器：

```
air.Localizer.localizer.addEventListener(air.Localizer.FILE_NOT_FOUND, fnfHandler);
air.Localizer.localizer.getFile("missing.html", null, "fr");
function fnfHandler(event)
{
    alert(event.fileName + ": " + event.locale);
}
```

**更多帮助主题**

构建基于 HTML 的多语言应用程序

# 第 21 章 : 路径环境变量

AIR SDK 包含几个可以从命令行或终端进行启动的程序。如果路径环境变量中包括 SDK bin 目录的路径，运行这些程序通常可以更加方便。

此处所列的信息讨论如何在 Windows、Mac 和 Linux 上设置路径，应该可以作为指南提供便利。但是，计算机配置彼此差异很大，因此该过程并非对每个系统都起作用。在这些情况下，您应能够从操作系统文档或 Internet 中查找必要的信息。

## 使用 Bash shell 在 Linux 和 Mac OS 上设置路径

当您在终端窗口中键入命令时， shell （即读取所键入内容并尝试做出适当响应的程序）必须首先在文件系统上找到命令程序。shell 会在名为 $PATH 的环境变量中所存储的目录列表中查找命令。若要查看路径中当前所列出的内容，请键入：

```
echo $PATH
```

这会返回以冒号分隔的目录列表，看上去应类似于：

```
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin
```

这样做的目的是将 AIR SDK bin 目录路径添加到列表中，以便 shell 可以找到 ADT 和 ADL 工具。假设已将 AIR SDK 放在 /Users/fred/SDKs/AIR 下，则通过下列命令可将必要的目录添加到该路径：

```
export PATH=$PATH:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/tools
```

注：如果您的路径包含空格字符，请使用反斜杠进行转义，如下所示：

```
/Users/fred\ jones/SDKs/AIR\ 2.5\ SDK/bin
```

您可以再次使用 echo 命令以确保它已工作：

```
echo $PATH
/usr/bin:/bin:/usr/sbin:/usr/local/bin:/usr/x11/bin:/Users/fred/SDKs/AIR/bin:/Users/fred/SDKs/android/too
ls
```

到目前为止一切顺利。您现在应该能够键入以下命令并取得令人鼓舞的响应：

```
adt -version
```

如果已正确修改 $PATH 变量，则该命令应报告 ADT 的版本。

但是，仍然存在一个问题；当下次启动新的终端窗口时，您会发现在该位置已找不到路径中的新条目。每次启动新的终端时都必须运行用于设置路径的命令。

此问题的常见解决方法是将该命令添加到 shell 所使用的其中一个启动脚本中。在 Mac OS 上，您可以在 ~/username 目录中创建 .bash_profile 文件，并且每次打开新的终端窗口时该文件都会运行。在 Ubuntu 上，启动新的终端窗口时运行的启动脚本是 .bashrc。其他 Linux 分发和 shell 程序具有类似的惯例。

若要将命令添加到 shell 启动脚本：

1 更改为您的主目录：

```
cd
```

2 创建 shell 配置文件（如有必要），并且通过 "cat >>" 将您所键入的文本重定向到文件结尾。针对您的操作系统和 shell 使用适当的文件。例如，您可以在 Mac OS 上使用 .bash_profile，而在 Ubuntu 上使用 .bashrc。

```
cat >> .bash_profile
```

3 键入要添加到文件的文本：

```
export PATH=$PATH:/Users/cward/SDKs/android/tools:/Users/cward/SDKs/AIR/bin
```

4 通过按下键盘上的 CTRL-SHIFT-D 结束文本重定向。

5 显示文件以确保一切都没问题：

```
cat .bash_profile
```

6 打开一个新的终端窗口以检查路径：

```
echo $PATH
```

应列出路径添加项。

如果稍后在其他目录中创建其中一个 SDK 的新版本，请务必更新配置文件中的路径命令。否则， shell 将继续使用旧版本。

# 在 Windows 上设置路径

当在 Windows 上打开命令窗口时，该窗口会继承在系统属性中所定义的全局环境变量。其中一个重要变量是路径，它是在键入要运行的程序名称时命令程序搜索的目录列表。使用命令窗口时若要查看路径中当前所包括的内容，可以键入：

```
set path
```

这将显示以分号分隔的目录列表，看上去类似于：

```
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
```

这样做的目的是将 AIR SDK bin 目录路径添加到列表中，以便命令程序可以找到 ADT 和 ADL 工具。假设已将 AIR SDK 放在 C:\SDKs\AIR 下，则可以通过以下过程添加正确的路径条目：

1 从 " 控制面板 " 或通过右键单击 " 我的电脑 " 图标并从菜单中选择 " 属性 " 来打开 " 系统属性 " 对话框。

2 在 " 高级 " 选项卡下，单击 " 环境变量 " 按钮。

3 在 " 环境变量 " 对话框的 " 系统变量 " 部分中选择路径条目

4 单击 " 编辑 "。

5 滚动到 " 变量值 " 字段中文本的结尾。

6 在当前值的最末端输入以下文本：

```
;C:\SDKs\AIR\bin
```

7 在所有对话框中单击 " 确定 " 以保存路径。

如果已打开任何命令窗口，您应意识到它们的环境不会更新。打开新的命令窗口并键入以下命令，以确保正确设置这些路径：

```
adt -version
```

如果稍后更改 AIR SDK 的位置或添加新的版本，请记得更新路径变量。